Module Information: IMAT1704 – Programming in Python – Portfolio

Group Logo:

Tecbook

Group Information:
P2785659
P2796362
P2724555
P2836714

Date: 03/02/2024

Table of Contents

1 – Introduction

This report is going to give a viewpoint on the design, implementation and user experience on our address book application. It will continue farther to explore system architectural details, source code as well as user guidance but with concentrate on GUI and object oriented ideas. The user guide leads you from usability, through evaluation of implementation, to design of user experience with the use of rich pictures. Following suggestions from our team members, we discuss upgrades related updates for programs, strategy creating for user involvement, and examining prospects for advertisement. Finally, we will include a conclusion which summarises our key findings.

2 – System Documentation

This component contains an architectural choice on software documentation, features of the programme Address Book via descriptions and language, source code structure and testing.

2.1 – Design Decisions and Architecture Descriptions

We developed our Address Book architecture on simplicity, usefulness, flexibility, and the system's driving philosophy, or 'trade-off' Using object-oriented programming with GUI components, such as the Tkinter library for GUIs and the PIL package for image processing, improves user experience by making the application easy and visually attractive.

Source code is modularly organised into four distinct files in one primary architectural design. These primary modules improve code readability, maintainability, and collective development. The system is organised around these architectural building blocks:

Contact Class: This module contains a contact's first and last name, address, mobile number, secondary phone number, email address, and optional photo. Slots enforce code robustness by allowing attribute constraints.

ContactEntryDialog Class: This is the class which shows user involvement by giving a nice interface for adding or editing of contact particulars on the address list. The layout of its user interface indicates ease of use and delivers a pleasant experience to the user.

AddressBookApp Class: This is the important and a driving class of the application that all key features are coordinated from. It holds the contact list, UI components, and interaction structure. The AddressBookApp facilitates contact adding, viewing, editing, and deletion, as well as sorting.

Also, as usual, the code is simply and brilliantly modularized, separated among four files – main.py, contact.py, contact_entry_dialog.py, and address_book_app.py. This does not only fit well with the best practices of software development, but it also goes a long way to guarantee the programme is readily maintenance and scalable. Each file is wrapped inside a distinct implementation of the application having in mind comfort while working together within the team, as well as being able to handle the codes on his or her side of the task.

In other words, the architectural decisions lean towards harmonic integration between simplicity, utility, and flexibility designed to acquire a highly strong, not forgetting user-friendly Address Book application.

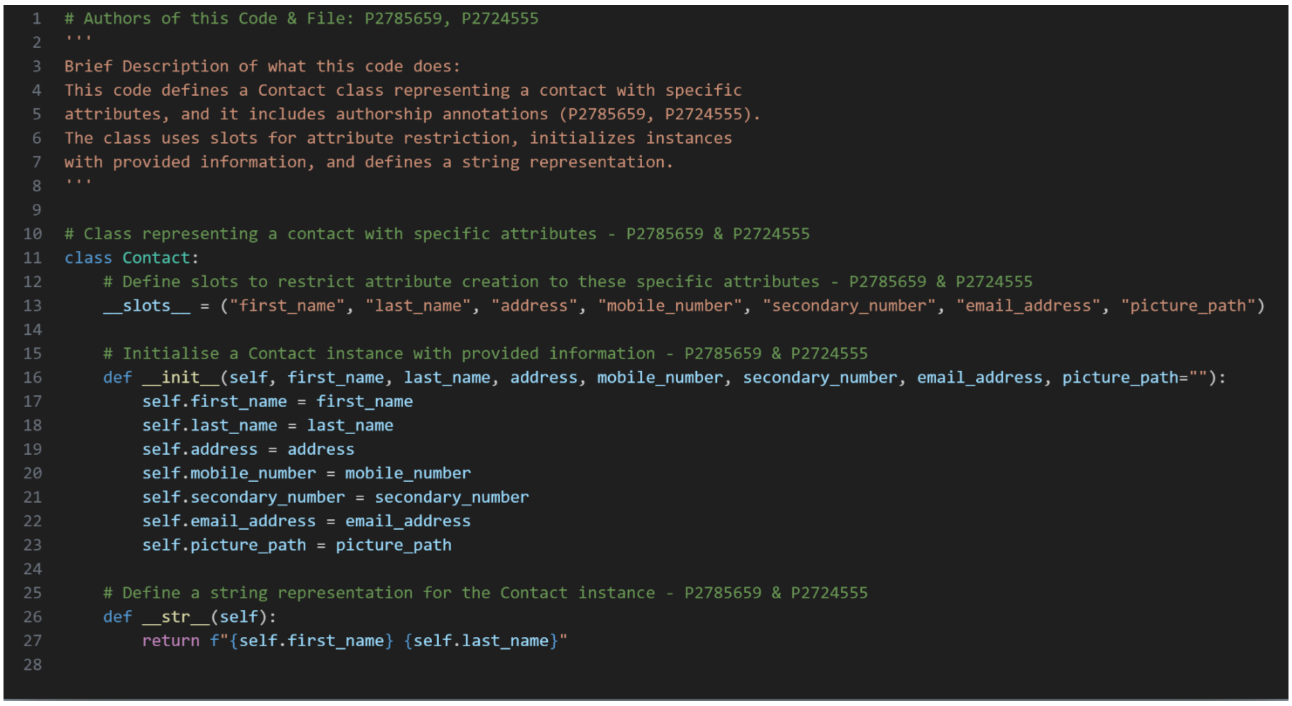
2.2 – Source Code

Source code is separated in four different files for the simplicity of reading, maintainability, and modularity. This part will provide a full explanation of the important classes and features created in the codebase alongside that I will also implement the screenshots for each, and every class mentioned. For the screenshots, I will be using an application called CodeSnap which is available through VS Code. VS Code was used to make this application.

2.2.1 – Contact Class (File: Contact.py)

The class Contact captures the fundamental structure of an address book contact. Through the usage of **slots** method in Python, this class diligently confines attribute constructions on the particular properties which include first and last names, address, mobile number, secondary number, email address and an optional picture URL. This attribute restriction assures a regulated and correctly formed one without the insertion of any superfluous attributes by accident.

The class features a nicely built **init** function that manages the seamless initialization of instances of contact. In this manner, it permits the contact instantiation with the supplied information, therefore encouraging a leaner and cleaner approach. Additionally, the **str** function enables for a human-readable string representation creation, improving the class's utility. This functionality becomes important during the contact with the instances of the class Contact, making this experience more informative and user-friendly. Here below is the screenshots of the code:

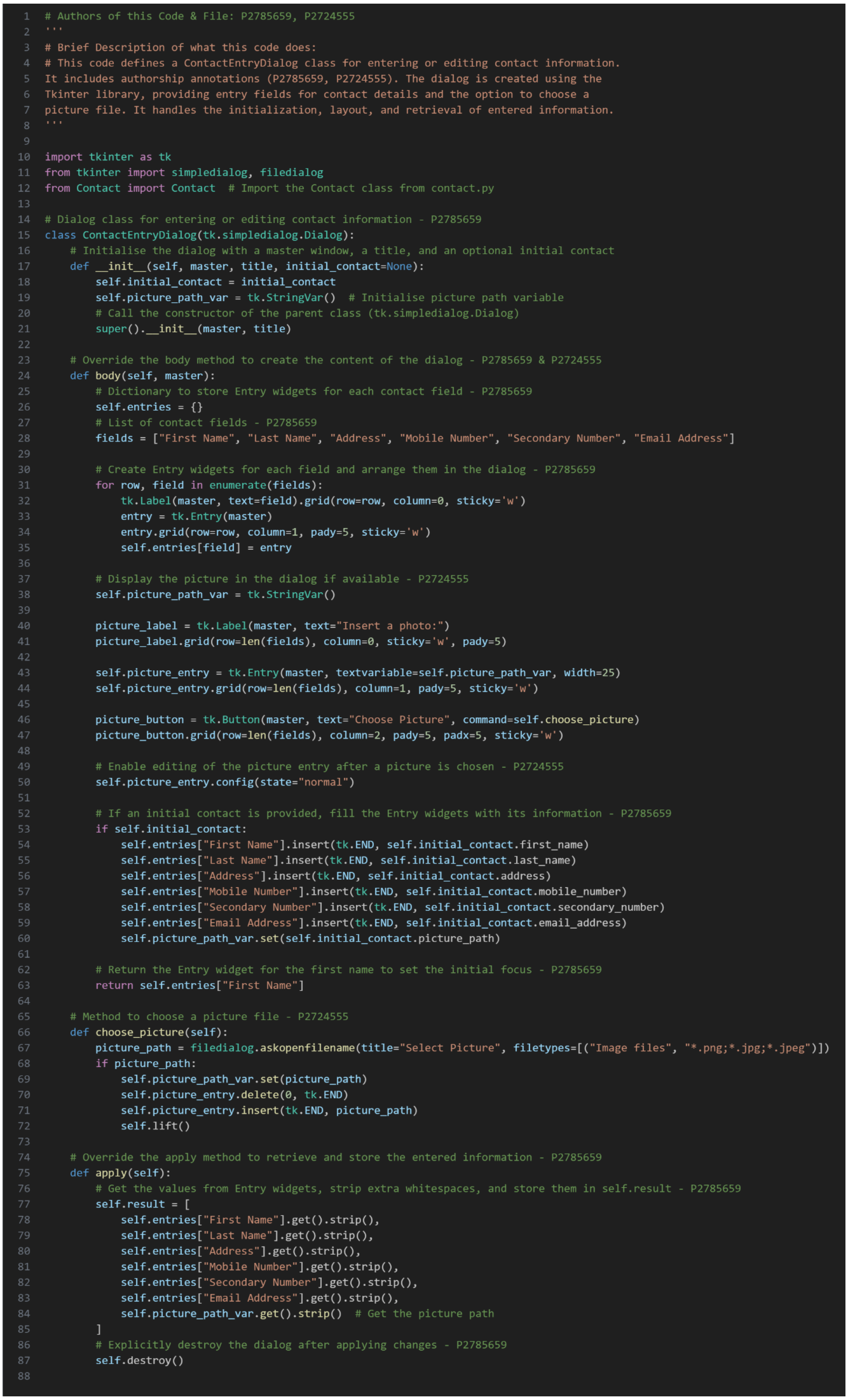


```python
# Authors of this Code & File: P2785659, P2724555
'''
Brief Description of what this code does:
This code defines a Contact class representing a contact with specific
attributes, and it includes authorship annotations (P2785659, P2724555).
The class uses slots for attribute restriction, initializes instances
with provided information, and defines a string representation.
'''

# Class representing a contact with specific attributes - P2785659 & P2724555
class Contact:
    # Define slots to restrict attribute creation to these specific attributes - P2785659 & P2724555
    __slots__ = ("first_name", "last_name", "address", "mobile_number", "secondary_number", "email_address", "picture_path")

    # Initialise a Contact instance with provided information - P2785659 & P2724555
    def __init__(self, first_name, last_name, address, mobile_number, secondary_number, email_address, picture_path=""):
        self.first_name = first_name
        self.last_name = last_name
        self.address = address
        self.mobile_number = mobile_number
        self.secondary_number = secondary_number
        self.email_address = email_address
        self.picture_path = picture_path

    # Define a string representation for the Contact instance - P2785659 & P2724555
    def __str__(self):
        return f"{self.first_name} {self.last_name}"

```

2.2.2 – ContactEntryDialog Class (File: contact_entry_dialog.py)

Simple and interactive insertion or modification of a contact has been added into the ContactEntryDialog class in the tkinter.simpledialog module. The dialog is visual with fields for all sorts of contacts, making it extremely simple to use and a highly engaging environment. One item that was incorporated is the capability that enables users to input their photo as part of the contacts.

All the setup, formatting, and retrieval of the input information is taken care of in this class in the most competent manner. The usage of Tkinter library helps in providing this a very consistent and beautiful appearance and feel. Leveraging simpledialog module's characteristics, the ContactEntryDialog class optimises the entire user experience to handle contact data and visually pleasant imitating next following and easy to comprehend procedure. Below is the code screenshot:

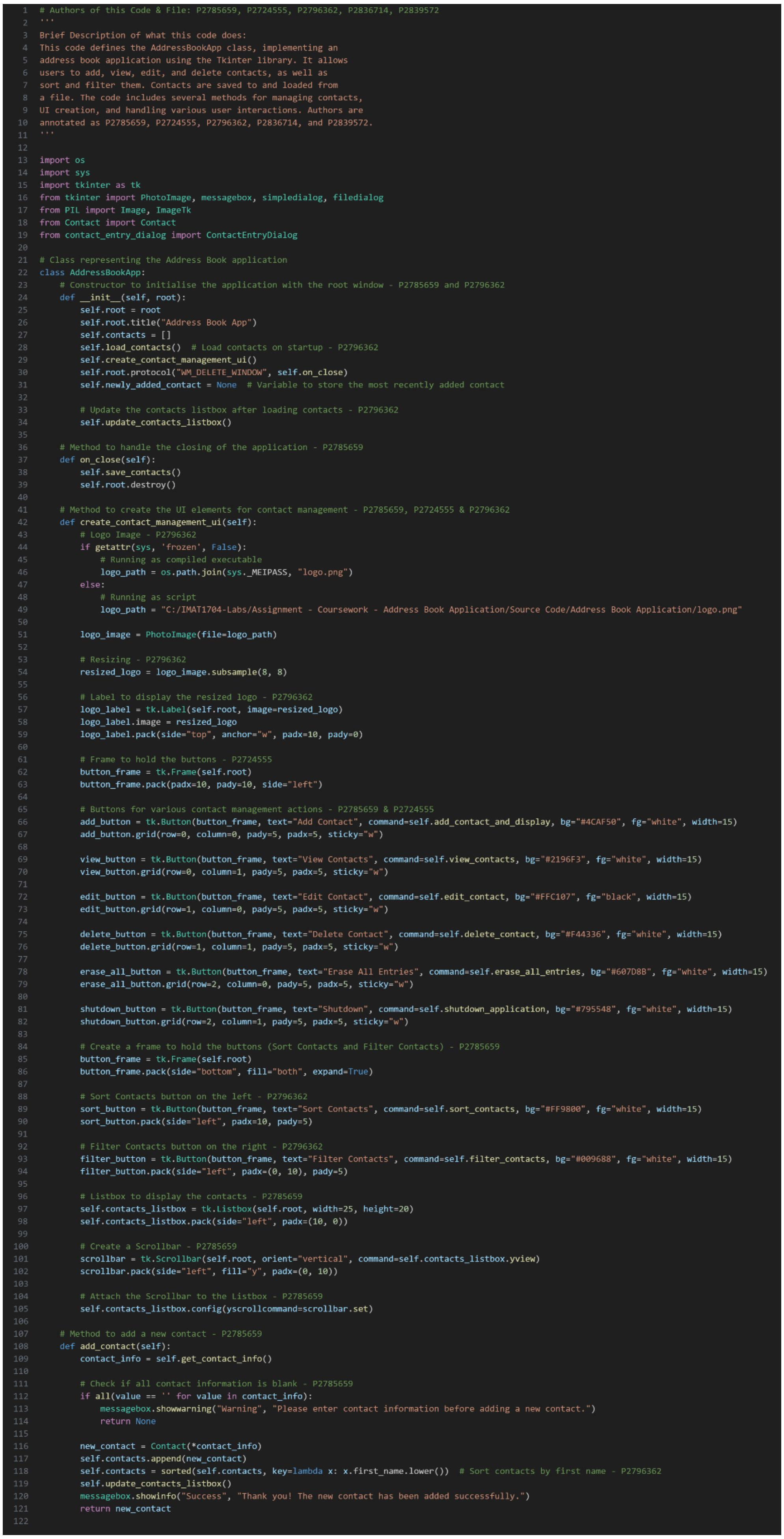


```python
# Authors of this Code & File: P2785659, P2724555
'''
# Brief Description of what this code does:
# This code defines a ContactEntryDialog class for entering or editing contact information.
It includes authorship annotations (P2785659, P2724555). The dialog is created using the
Tkinter library, providing entry fields for contact details and the option to choose a
picture file. It handles the initialization, layout, and retrieval of entered information.
'''

import tkinter as tk
from tkinter import simpledialog, filedialog
from Contact import Contact  # Import the Contact class from contact.py

# Dialog class for entering or editing contact information - P2785659
class ContactEntryDialog(tk.simpledialog.Dialog):
    # Initialise the dialog with a master window, a title, and an optional initial contact
    def __init__(self, master, title, initial_contact=None):
        self.initial_contact = initial_contact
        self.picture_path_var = tk.StringVar()  # Initialise picture path variable
        # Call the constructor of the parent class (tk.simpledialog.Dialog)
        super().__init__(master, title)

    # Override the body method to create the content of the dialog - P2785659 & P2724555
    def body(self, master):
        # Dictionary to store Entry widgets for each contact field - P2785659
        self.entries = {}
        # List of contact fields - P2785659
        fields = ["First Name", "Last Name", "Address", "Mobile Number", "Secondary Number", "Email Address"]

        # Create Entry widgets for each field and arrange them in the dialog - P2785659
        for row, field in enumerate(fields):
            tk.Label(master, text=field).grid(row=row, column=0, sticky='w')
            entry = tk.Entry(master)
            entry.grid(row=row, column=1, pady=5, sticky='w')
            self.entries[field] = entry

        # Display the picture in the dialog if available - P2724555
        self.picture_path_var = tk.StringVar()

        picture_label = tk.Label(master, text="Insert a photo:")
        picture_label.grid(row=len(fields), column=0, sticky='w', pady=5)

        self.picture_entry = tk.Entry(master, textvariable=self.picture_path_var, width=25)
        self.picture_entry.grid(row=len(fields), column=1, pady=5, sticky='w')

        picture_button = tk.Button(master, text="Choose Picture", command=self.choose_picture)
        picture_button.grid(row=len(fields), column=2, pady=5, padx=5, sticky='w')

        # Enable editing of the picture entry after a picture is chosen - P2724555
        self.picture_entry.config(state="normal")

        # If an initial contact is provided, fill the Entry widgets with its information - P2785659
        if self.initial_contact:
            self.entries["First Name"].insert(tk.END, self.initial_contact.first_name)
            self.entries["Last Name"].insert(tk.END, self.initial_contact.last_name)
            self.entries["Address"].insert(tk.END, self.initial_contact.address)
            self.entries["Mobile Number"].insert(tk.END, self.initial_contact.mobile_number)
            self.entries["Secondary Number"].insert(tk.END, self.initial_contact.secondary_number)
            self.entries["Email Address"].insert(tk.END, self.initial_contact.email_address)
            self.picture_path_var.set(self.initial_contact.picture_path)

        # Return the Entry widget for the first name to set the initial focus - P2785659
        return self.entries["First Name"]

    # Method to choose a picture file - P2724555
    def choose_picture(self):
        picture_path = filedialog.askopenfilename(title="Select Picture", filetypes=[("Image files", "*.png;*.jpg;*.jpeg")])
        if picture_path:
            self.picture_path_var.set(picture_path)
            self.picture_entry.delete(0, tk.END)
            self.picture_entry.insert(tk.END, picture_path)
            self.lift()

    # Override the apply method to retrieve and store the entered information - P2785659
    def apply(self):
        # Get the values from Entry widgets, strip extra whitespaces, and store them in self.result - P2785659
        self.result = [
            self.entries["First Name"].get().strip(),
            self.entries["Last Name"].get().strip(),
            self.entries["Address"].get().strip(),
            self.entries["Mobile Number"].get().strip(),
            self.entries["Secondary Number"].get().strip(),
            self.entries["Email Address"].get().strip(),
            self.picture_path_var.get().strip()  # Get the picture path
        ]
        # Explicitly destroy the dialog after applying changes - P2785659
        self.destroy()
```

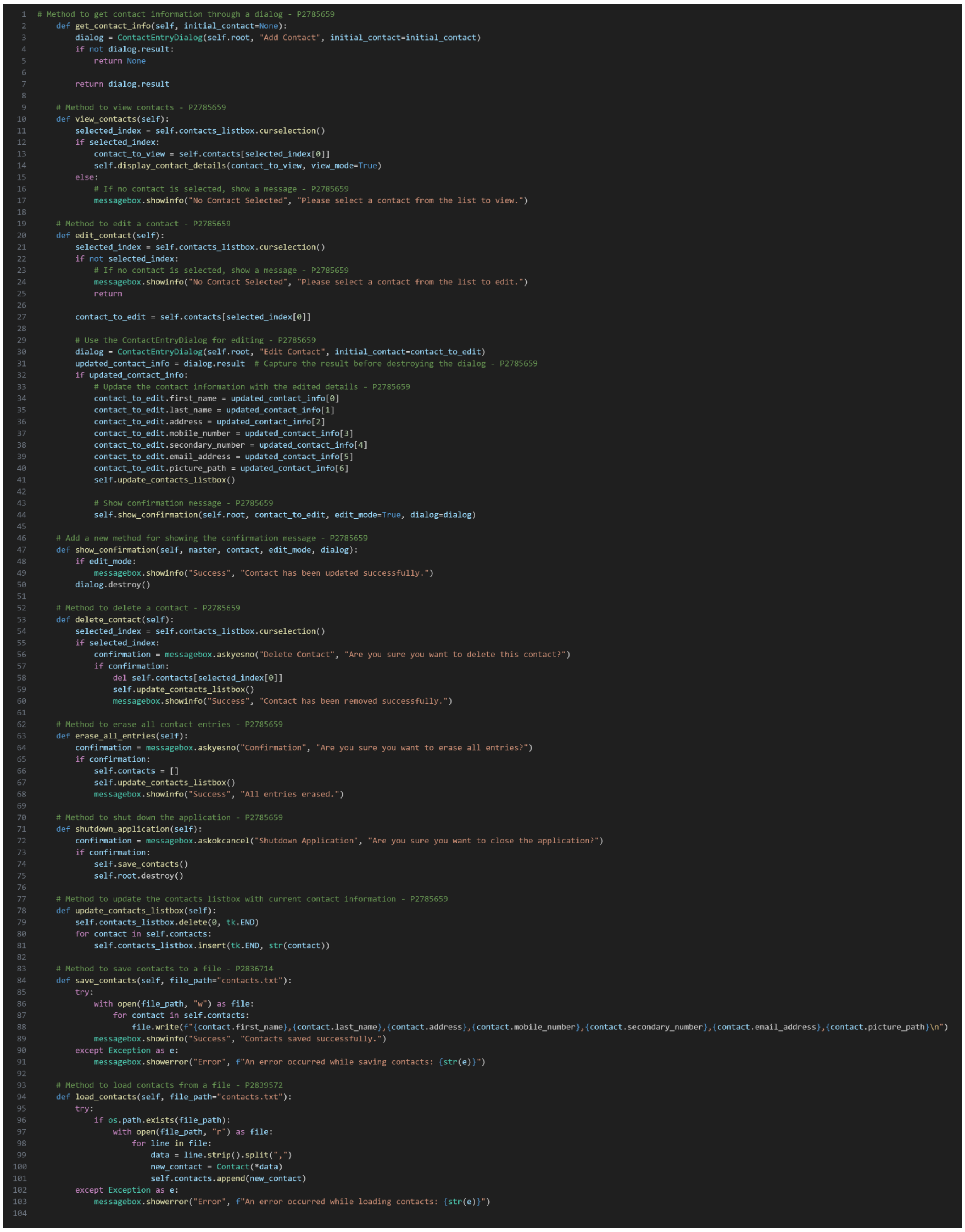
2.2.3 – AddressBookApp Class (File: address_book_app.py)

The AddressBookApp class works as the backbone of the address book application, organising a huge number of functionality in an organised method. Designed using Tkinter, it allows the adding, viewing, editing, and removal of contacts by the user with simplicity. Meanwhile, its capacity in sorting, filtering or storing the acquired data from a user further palpitates the concentrated accomplishments inside the programme.

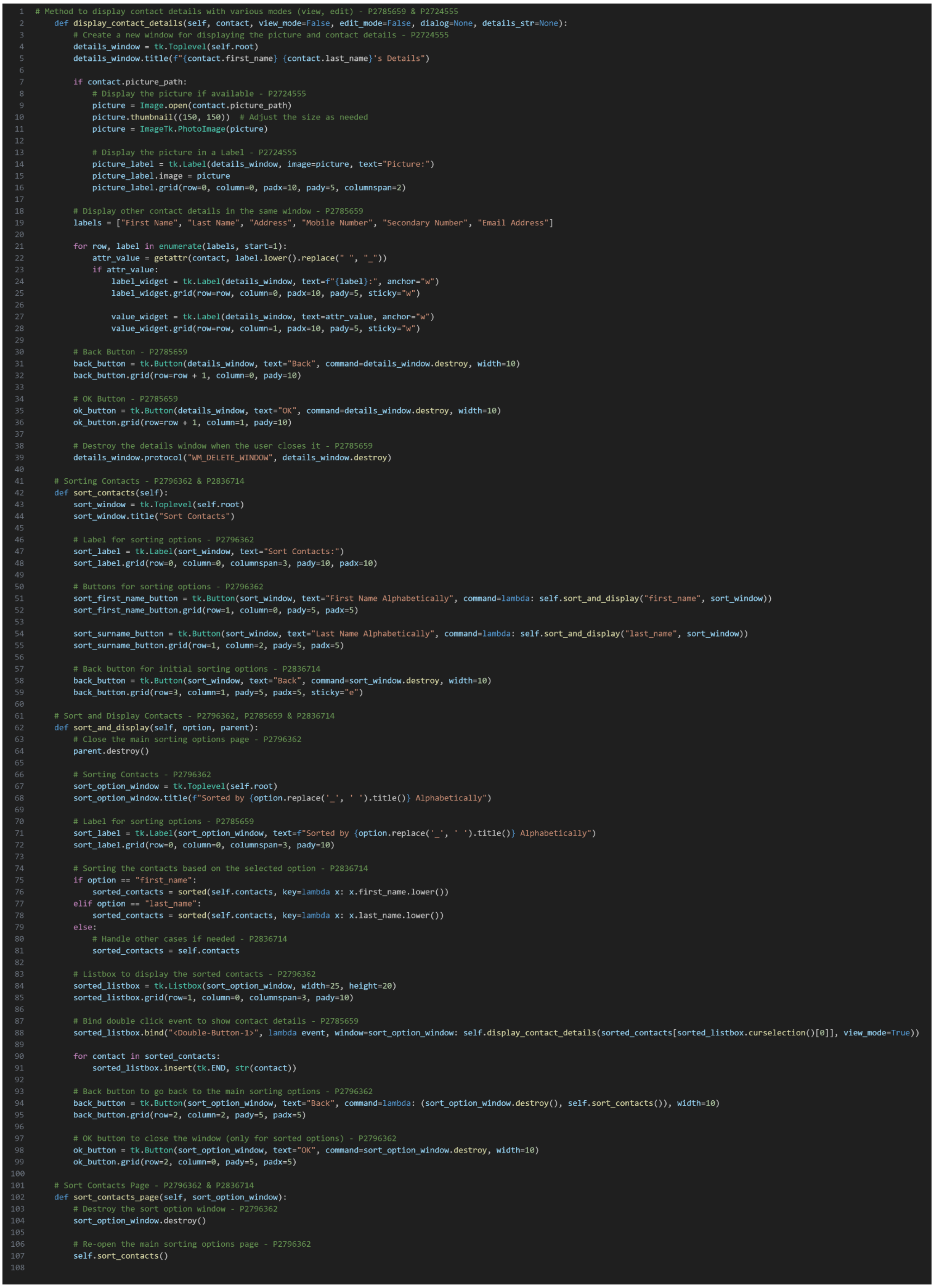
With structured classes fronted by AddressBookApp and reinforced with utility classes Contact and ContactEntryDialog, the construction implementation is done under an object-oriented architecture. This design decision with all the object-oriented programming concepts enables greater structure and readability of the code. Together with the object-oriented approach, the seamless connecting of the utility classes together with the primary application environment represents a type of modular and extensible design of the project that permits its long-term maintainability and scalability. Screenshots of the code as below:



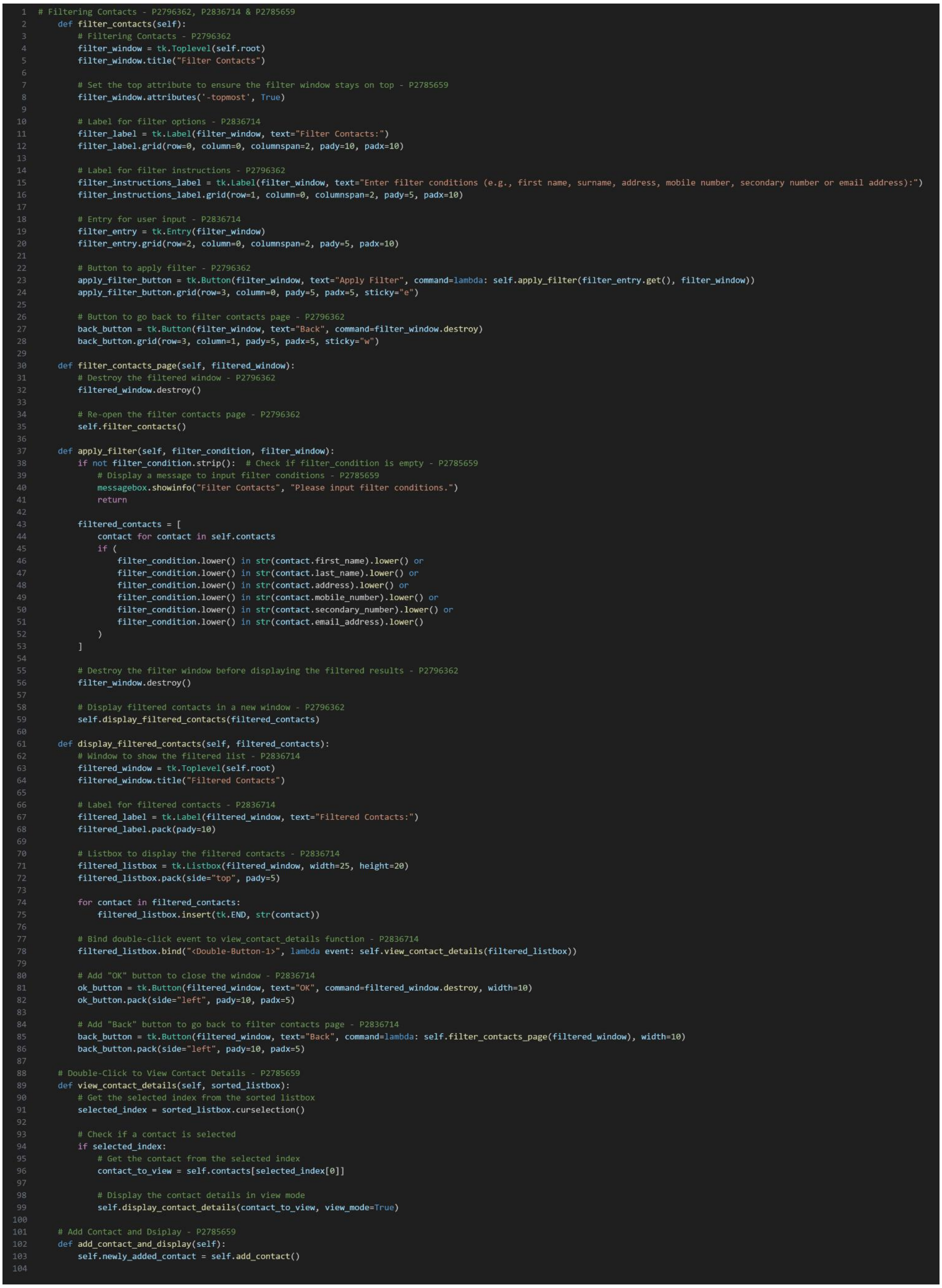
```python
# Authors of this Code & File: P2785659, P2724555, P2796362, P2836714, P2839572
'''
Brief Description of what this code does:
This code defines the AddressBookApp class, implementing an
address book application using the Tkinter library. It allows
users to add, view, edit, and delete contacts, as well as
sort and filter them. Contacts are saved to and loaded from
a file. The code includes several methods for managing contacts,
UI creation, and handling various user interactions. Authors are
annotated as P2785659, P2724555, P2796362, P2836714, and P2839572.
'''

import os
import sys
import tkinter as tk
from tkinter import PhotoImage, messagebox, simpledialog, filedialog
from PIL import Image, ImageTk
from Contact import Contact
from contact_entry_dialog import ContactEntryDialog

# Class representing the Address Book application
class AddressBookApp:
    # Constructor to initialise the application with the root window - P2785659 and P2796362
    def __init__(self, root):
        self.root = root
        self.root.title("Address Book App")
        self.contacts = []
        self.load_contacts()  # Load contacts on startup - P2796362
        self.create_contact_management_ui()
        self.root.protocol("WM_DELETE_WINDOW", self.on_close)
        self.newly_added_contact = None  # Variable to store the most recently added contact

        # Update the contacts listbox after loading contacts - P2796362
        self.update_contacts_listbox()

    # Method to handle the closing of the application - P2785659
    def on_close(self):
        self.save_contacts()
        self.root.destroy()

    # Method to create the UI elements for contact management - P2785659, P2724555 & P2796362
    def create_contact_management_ui(self):
        # Logo Image - P2796362
        if getattr(sys, 'frozen', False):
            # Running as compiled executable
            logo_path = os.path.join(sys._MEIPASS, "logo.png")
        else:
            # Running as script
            logo_path = "C:/IMAT1704-Labs/Assignment - Coursework - Address Book Application/Source Code/Address Book Application/logo.png"

        logo_image = PhotoImage(file=logo_path)

        # Resizing - P2796362
        resized_logo = logo_image.subsample(8, 8)

        # Label to display the resized logo - P2796362
        logo_label = tk.Label(self.root, image=resized_logo)
        logo_label.image = resized_logo
        logo_label.pack(side="top", anchor="w", padx=10, pady=0)

        # Frame to hold the buttons - P2724555
        button_frame = tk.Frame(self.root)
        button_frame.pack(padx=10, pady=10, side="left")

        # Buttons for various contact management actions - P2785659 & P2724555
        add_button = tk.Button(button_frame, text="Add Contact", command=self.add_contact_and_display, bg="#4CAF50", fg="white", width=15)
        add_button.grid(row=0, column=0, pady=5, padx=5, sticky="w")

        view_button = tk.Button(button_frame, text="View Contacts", command=self.view_contacts, bg="#2196F3", fg="white", width=15)
        view_button.grid(row=0, column=1, pady=5, padx=5, sticky="w")

        edit_button = tk.Button(button_frame, text="Edit Contact", command=self.edit_contact, bg="#FFC107", fg="black", width=15)
        edit_button.grid(row=1, column=0, pady=5, padx=5, sticky="w")

        delete_button = tk.Button(button_frame, text="Delete Contact", command=self.delete_contact, bg="#F44336", fg="white", width=15)
        delete_button.grid(row=1, column=1, pady=5, padx=5, sticky="w")

        erase_all_button = tk.Button(button_frame, text="Erase All Entries", command=self.erase_all_entries, bg="#607D8B", fg="white", width=15)
        erase_all_button.grid(row=2, column=0, pady=5, padx=5, sticky="w")

        shutdown_button = tk.Button(button_frame, text="Shutdown", command=self.shutdown_application, bg="#795548", fg="white", width=15)
        shutdown_button.grid(row=2, column=1, pady=5, padx=5, sticky="w")

        # Create a frame to hold the buttons (Sort Contacts and Filter Contacts) - P2785659
        button_frame = tk.Frame(self.root)
        button_frame.pack(side="bottom", fill="both", expand=True)

        # Sort Contacts button on the left - P2796362
        sort_button = tk.Button(button_frame, text="Sort Contacts", command=self.sort_contacts, bg="#FF9800", fg="white", width=15)
        sort_button.pack(side="left", padx=10, pady=5)

        # Filter Contacts button on the right - P2796362
        filter_button = tk.Button(button_frame, text="Filter Contacts", command=self.filter_contacts, bg="#009688", fg="white", width=15)
        filter_button.pack(side="left", padx=(0, 10), pady=5)

        # Listbox to display the contacts - P2785659
        self.contacts_listbox = tk.Listbox(self.root, width=25, height=20)
        self.contacts_listbox.pack(side="left", padx=(10, 0))

        # Create a Scrollbar - P2785659
        scrollbar = tk.Scrollbar(self.root, orient="vertical", command=self.contacts_listbox.yview)
        scrollbar.pack(side="left", fill="y", padx=(0, 10))

        # Attach the Scrollbar to the Listbox - P2785659
        self.contacts_listbox.config(yscrollcommand=scrollbar.set)

    # Method to add a new contact - P2785659
    def add_contact(self):
        contact_info = self.get_contact_info()

        # Check if all contact information is blank - P2785659
        if all(value == '' for value in contact_info):
            messagebox.showwarning("Warning", "Please enter contact information before adding a new contact.")
            return None

        new_contact = Contact(*contact_info)
        self.contacts.append(new_contact)
        self.contacts = sorted(self.contacts, key=lambda x: x.first_name.lower())  # Sort contacts by first name - P2796362
        self.update_contacts_listbox()
        messagebox.showinfo("Success", "Thank you! The new contact has been added successfully.")
        return new_contact

```



```python
# Method to get contact information through a dialog - P2785659
    def get_contact_info(self, initial_contact=None):
        dialog = ContactEntryDialog(self.root, "Add Contact", initial_contact=initial_contact)
        if not dialog.result:
            return None

        return dialog.result

    # Method to view contacts - P2785659
    def view_contacts(self):
        selected_index = self.contacts_listbox.curselection()
        if selected_index:
            contact_to_view = self.contacts[selected_index[0]]
            self.display_contact_details(contact_to_view, view_mode=True)
        else:
            # If no contact is selected, show a message - P2785659
            messagebox.showinfo("No Contact Selected", "Please select a contact from the list to view.")

    # Method to edit a contact - P2785659
    def edit_contact(self):
        selected_index = self.contacts_listbox.curselection()
        if not selected_index:
            # If no contact is selected, show a message - P2785659
            messagebox.showinfo("No Contact Selected", "Please select a contact from the list to edit.")
            return

        contact_to_edit = self.contacts[selected_index[0]]

        # Use the ContactEntryDialog for editing - P2785659
        dialog = ContactEntryDialog(self.root, "Edit Contact", initial_contact=contact_to_edit)
        updated_contact_info = dialog.result  # Capture the result before destroying the dialog - P2785659
        if updated_contact_info:
            # Update the contact information with the edited details - P2785659
            contact_to_edit.first_name = updated_contact_info[0]
            contact_to_edit.last_name = updated_contact_info[1]
            contact_to_edit.address = updated_contact_info[2]
            contact_to_edit.mobile_number = updated_contact_info[3]
            contact_to_edit.secondary_number = updated_contact_info[4]
            contact_to_edit.email_address = updated_contact_info[5]
            contact_to_edit.picture_path = updated_contact_info[6]
            self.update_contacts_listbox()

            # Show confirmation message - P2785659
            self.show_confirmation(self.root, contact_to_edit, edit_mode=True, dialog=dialog)

    # Add a new method for showing the confirmation message - P2785659
    def show_confirmation(self, master, contact, edit_mode, dialog):
        if edit_mode:
            messagebox.showinfo("Success", "Contact has been updated successfully.")
        dialog.destroy()

    # Method to delete a contact - P2785659
    def delete_contact(self):
        selected_index = self.contacts_listbox.curselection()
        if selected_index:
            confirmation = messagebox.askyesno("Delete Contact", "Are you sure you want to delete this contact?")
            if confirmation:
                del self.contacts[selected_index[0]]
                self.update_contacts_listbox()
                messagebox.showinfo("Success", "Contact has been removed successfully.")

    # Method to erase all contact entries - P2785659
    def erase_all_entries(self):
        confirmation = messagebox.askyesno("Confirmation", "Are you sure you want to erase all entries?")
        if confirmation:
            self.contacts = []
            self.update_contacts_listbox()
            messagebox.showinfo("Success", "All entries erased.")

    # Method to shut down the application - P2785659
    def shutdown_application(self):
        confirmation = messagebox.askokcancel("Shutdown Application", "Are you sure you want to close the application?")
        if confirmation:
            self.save_contacts()
            self.root.destroy()

    # Method to update the contacts listbox with current contact information - P2785659
    def update_contacts_listbox(self):
        self.contacts_listbox.delete(0, tk.END)
        for contact in self.contacts:
            self.contacts_listbox.insert(tk.END, str(contact))

    # Method to save contacts to a file - P2836714
    def save_contacts(self, file_path="contacts.txt"):
        try:
            with open(file_path, "w") as file:
                for contact in self.contacts:
                    file.write(f"{contact.first_name},{contact.last_name},{contact.address},{contact.mobile_number},{contact.secondary_number},{contact.email_address},{contact.picture_path}\n")
            messagebox.showinfo("Success", "Contacts saved successfully.")
        except Exception as e:
            messagebox.showerror("Error", f"An error occurred while saving contacts: {str(e)}")

    # Method to load contacts from a file - P2839572
    def load_contacts(self, file_path="contacts.txt"):
        try:
            if os.path.exists(file_path):
                with open(file_path, "r") as file:
                    for line in file:
                        data = line.strip().split(",")
                        new_contact = Contact(*data)
                        self.contacts.append(new_contact)
        except Exception as e:
            messagebox.showerror("Error", f"An error occurred while loading contacts: {str(e)}")

```



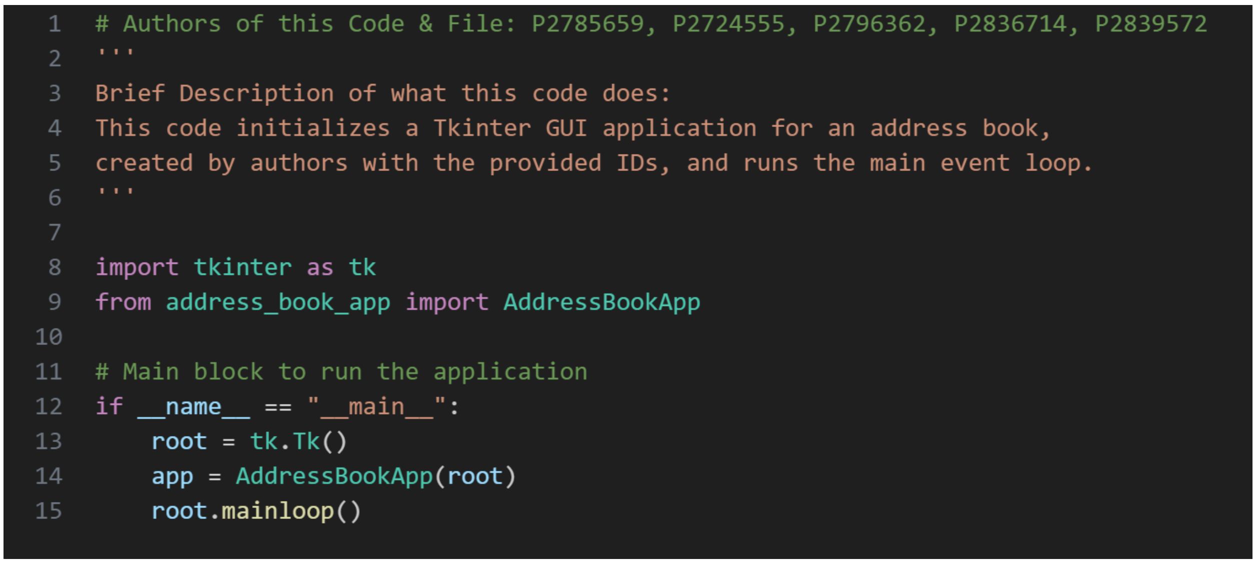
```
# Method to display contact details with various modes (view, edit) - P2785659 & P2724555
    def display_contact_details(self, contact, view_mode=False, edit_mode=False, dialog=None, details_str=None):
        # Create a new window for displaying the picture and contact details - P2724555
        details_window = tk.Toplevel(self.root)
        details_window.title(f"{contact.first_name} {contact.last_name}'s Details")

        if contact.picture_path:
            # Display the picture if available - P2724555
            picture = Image.open(contact.picture_path)
            picture.thumbnail((150, 150))  # Adjust the size as needed
            picture = ImageTk.PhotoImage(picture)

            # Display the picture in a Label - P2724555
            picture_label = tk.Label(details_window, image=picture, text="Picture:")
            picture_label.image = picture
            picture_label.grid(row=0, column=0, padx=10, pady=5, columnspan=2)

        # Display other contact details in the same window - P2785659
        labels = ["First Name", "Last Name", "Address", "Mobile Number", "Secondary Number", "Email Address"]

        for row, label in enumerate(labels, start=1):
            attr_value = getattr(contact, label.lower().replace(" ", "_"))
            if attr_value:
                label_widget = tk.Label(details_window, text=f"{label}:", anchor="w")
                label_widget.grid(row=row, column=0, padx=10, pady=5, sticky="w")

                value_widget = tk.Label(details_window, text=attr_value, anchor="w")
                value_widget.grid(row=row, column=1, padx=10, pady=5, sticky="w")

        # Back Button - P2785659
        back_button = tk.Button(details_window, text="Back", command=details_window.destroy, width=10)
        back_button.grid(row=row + 1, column=0, pady=10)

        # OK Button - P2785659
        ok_button = tk.Button(details_window, text="OK", command=details_window.destroy, width=10)
        ok_button.grid(row=row + 1, column=1, pady=10)

        # Destroy the details window when the user closes it - P2785659
        details_window.protocol("WM_DELETE_WINDOW", details_window.destroy)

    # Sorting Contacts - P2796362 & P2836714
    def sort_contacts(self):
        sort_window = tk.Toplevel(self.root)
        sort_window.title("Sort Contacts")

        # Label for sorting options - P2796362
        sort_label = tk.Label(sort_window, text="Sort Contacts:")
        sort_label.grid(row=0, column=0, columnspan=3, pady=10, padx=10)

        # Buttons for sorting options - P2796362
        sort_first_name_button = tk.Button(sort_window, text="First Name Alphabetically", command=lambda: self.sort_and_display("first_name", sort_window))
        sort_first_name_button.grid(row=1, column=0, pady=5, padx=5)

        sort_surname_button = tk.Button(sort_window, text="Last Name Alphabetically", command=lambda: self.sort_and_display("last_name", sort_window))
        sort_surname_button.grid(row=1, column=2, pady=5, padx=5)

        # Back button for initial sorting options - P2836714
        back_button = tk.Button(sort_window, text="Back", command=sort_window.destroy, width=10)
        back_button.grid(row=3, column=1, pady=5, padx=5, sticky="e")

    # Sort and Display Contacts - P2796362, P2785659 & P2836714
    def sort_and_display(self, option, parent):
        # Close the main sorting options page - P2796362
        parent.destroy()

        # Sorting Contacts - P2796362
        sort_option_window = tk.Toplevel(self.root)
        sort_option_window.title(f"Sorted by {option.replace('_', ' ').title()} Alphabetically")

        # Label for sorting options - P2785659
        sort_label = tk.Label(sort_option_window, text=f"Sorted by {option.replace('_', ' ').title()} Alphabetically")
        sort_label.grid(row=0, column=0, columnspan=3, pady=10)

        # Sorting the contacts based on the selected option - P2836714
        if option == "first_name":
            sorted_contacts = sorted(self.contacts, key=lambda x: x.first_name.lower())
        elif option == "last_name":
            sorted_contacts = sorted(self.contacts, key=lambda x: x.last_name.lower())
        else:
            # Handle other cases if needed - P2836714
            sorted_contacts = self.contacts

        # Listbox to display the sorted contacts - P2796362
        sorted_listbox = tk.Listbox(sort_option_window, width=25, height=20)
        sorted_listbox.grid(row=1, column=0, columnspan=3, pady=10)

        # Bind double click event to show contact details - P2785659
        sorted_listbox.bind("<Double-Button-1>", lambda event, window=sort_option_window: self.display_contact_details(sorted_contacts[sorted_listbox.curselection()[0]], view_mode=True))

        for contact in sorted_contacts:
            sorted_listbox.insert(tk.END, str(contact))

        # Back button to go back to the main sorting options - P2796362
        back_button = tk.Button(sort_option_window, text="Back", command=lambda: (sort_option_window.destroy(), self.sort_contacts()), width=10)
        back_button.grid(row=2, column=2, pady=5, padx=5)

        # OK button to close the window (only for sorted options) - P2796362
        ok_button = tk.Button(sort_option_window, text="OK", command=sort_option_window.destroy, width=10)
        ok_button.grid(row=2, column=0, pady=5, padx=5)

    # Sort Contacts Page - P2796362 & P2836714
    def sort_contacts_page(self, sort_option_window):
        # Destroy the sort option window - P2796362
        sort_option_window.destroy()

        # Re-open the main sorting options page - P2796362
        self.sort_contacts()
```



```python
# Filtering Contacts - P2796362, P2836714 & P2785659
    def filter_contacts(self):
        # Filtering Contacts - P2796362
        filter_window = tk.Toplevel(self.root)
        filter_window.title("Filter Contacts")

        # Set the top attribute to ensure the filter window stays on top - P2785659
        filter_window.attributes('-topmost', True)

        # Label for filter options - P2836714
        filter_label = tk.Label(filter_window, text="Filter Contacts:")
        filter_label.grid(row=0, column=0, columnspan=2, pady=10, padx=10)

        # Label for filter instructions - P2796362
        filter_instructions_label = tk.Label(filter_window, text="Enter filter conditions (e.g., first name, surname, address, mobile number, secondary number or email address):")
        filter_instructions_label.grid(row=1, column=0, columnspan=2, pady=5, padx=10)

        # Entry for user input - P2836714
        filter_entry = tk.Entry(filter_window)
        filter_entry.grid(row=2, column=0, columnspan=2, pady=5, padx=10)

        # Button to apply filter - P2796362
        apply_filter_button = tk.Button(filter_window, text="Apply Filter", command=lambda: self.apply_filter(filter_entry.get(), filter_window))
        apply_filter_button.grid(row=3, column=0, pady=5, padx=5, sticky="e")

        # Button to go back to filter contacts page - P2796362
        back_button = tk.Button(filter_window, text="Back", command=filter_window.destroy)
        back_button.grid(row=3, column=1, pady=5, padx=5, sticky="w")

    def filter_contacts_page(self, filtered_window):
        # Destroy the filtered window - P2796362
        filtered_window.destroy()

        # Re-open the filter contacts page - P2796362
        self.filter_contacts()

    def apply_filter(self, filter_condition, filter_window):
        if not filter_condition.strip():  # Check if filter_condition is empty - P2785659
            # Display a message to input filter conditions - P2785659
            messagebox.showinfo("Filter Contacts", "Please input filter conditions.")
            return

        filtered_contacts = [
            contact for contact in self.contacts
            if (
                filter_condition.lower() in str(contact.first_name).lower() or
                filter_condition.lower() in str(contact.last_name).lower() or
                filter_condition.lower() in str(contact.address).lower() or
                filter_condition.lower() in str(contact.mobile_number).lower() or
                filter_condition.lower() in str(contact.secondary_number).lower() or
                filter_condition.lower() in str(contact.email_address).lower()
            )
        ]

        # Destroy the filter window before displaying the filtered results - P2796362
        filter_window.destroy()

        # Display filtered contacts in a new window - P2796362
        self.display_filtered_contacts(filtered_contacts)

    def display_filtered_contacts(self, filtered_contacts):
        # Window to show the filtered list - P2836714
        filtered_window = tk.Toplevel(self.root)
        filtered_window.title("Filtered Contacts")

        # Label for filtered contacts - P2836714
        filtered_label = tk.Label(filtered_window, text="Filtered Contacts:")
        filtered_label.pack(pady=10)

        # Listbox to display the filtered contacts - P2836714
        filtered_listbox = tk.Listbox(filtered_window, width=25, height=20)
        filtered_listbox.pack(side="top", pady=5)

        for contact in filtered_contacts:
            filtered_listbox.insert(tk.END, str(contact))

        # Bind double-click event to view_contact_details function - P2836714
        filtered_listbox.bind("<Double-Button-1>", lambda event: self.view_contact_details(filtered_listbox))

        # Add "OK" button to close the window - P2836714
        ok_button = tk.Button(filtered_window, text="OK", command=filtered_window.destroy, width=10)
        ok_button.pack(side="left", pady=10, padx=5)

        # Add "Back" button to go back to filter contacts page - P2836714
        back_button = tk.Button(filtered_window, text="Back", command=lambda: self.filter_contacts_page(filtered_window), width=10)
        back_button.pack(side="left", pady=10, padx=5)

    # Double-Click to View Contact Details - P2785659
    def view_contact_details(self, sorted_listbox):
        # Get the selected index from the sorted listbox
        selected_index = sorted_listbox.curselection()

        # Check if a contact is selected
        if selected_index:
            # Get the contact from the selected index
            contact_to_view = self.contacts[selected_index[0]]

            # Display the contact details in view mode
            self.display_contact_details(contact_to_view, view_mode=True)

    # Add Contact and Dsiplay - P2785659
    def add_contact_and_display(self):
        self.newly_added_contact = self.add_contact()

```

2.2.4 – Main Application Entry Point (File: main.py)

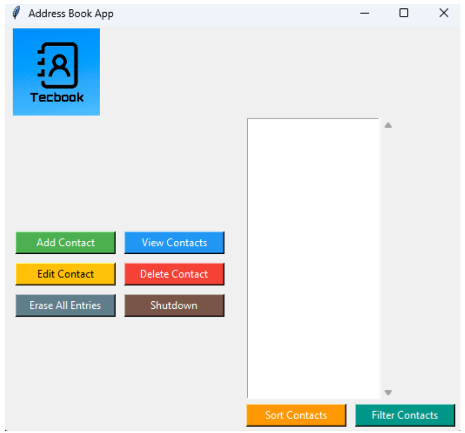
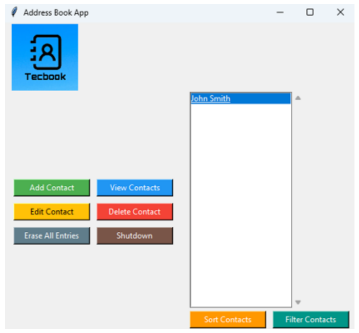
This is the entry point into the address book and the programme begins executing from here. Simply, it initializes a Tkinter GUI setting the scene for the remainder of the main event loop. This is basic but extremely crucial function exhibiting the beauty of the codebase that demonstrates conciseness, simplicity and efficiency. Here is the screenshot of the code:

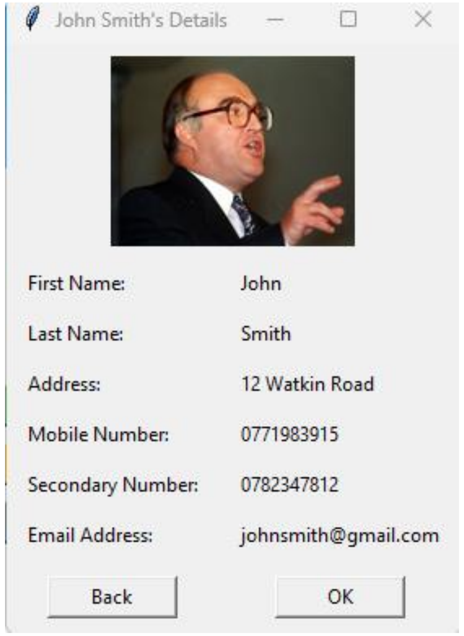


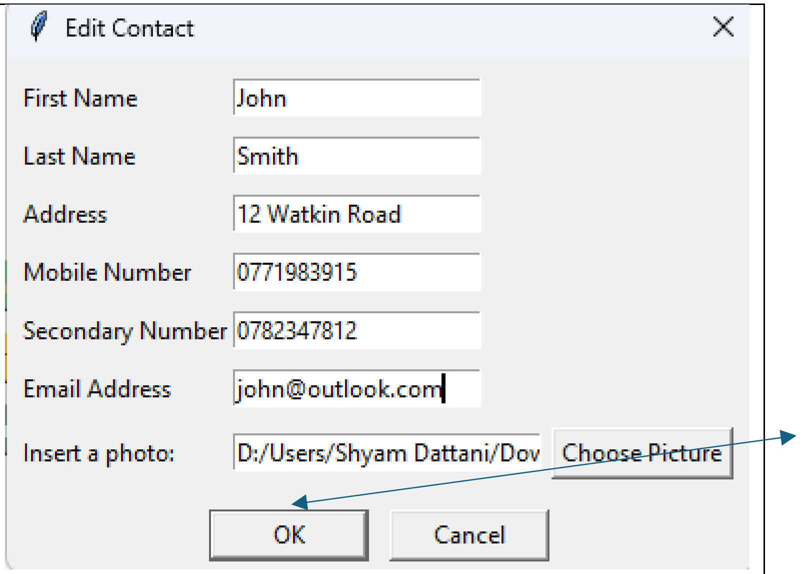
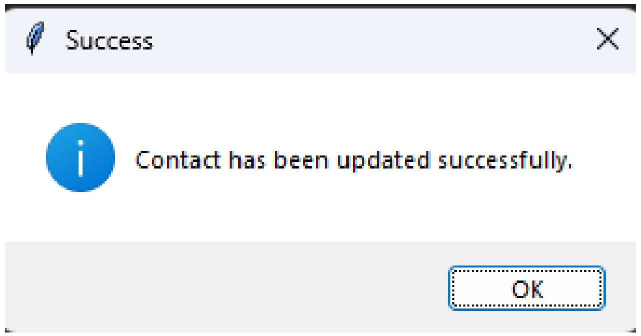
```python
# Authors of this Code & File: P2785659, P2724555, P2796362, P2836714, P2839572
'''
Brief Description of what this code does:
This code initializes a Tkinter GUI application for an address book,
created by authors with the provided IDs, and runs the main event loop.
'''

import tkinter as tk
from address_book_app import AddressBookApp

# Main block to run the application
if __name__ == "__main__":
    root = tk.Tk()
    app = AddressBookApp(root)
    root.mainloop()
```

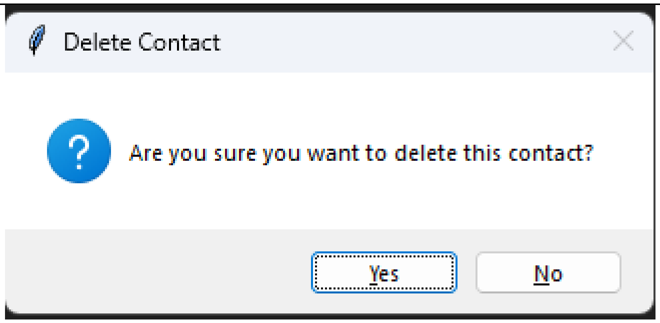
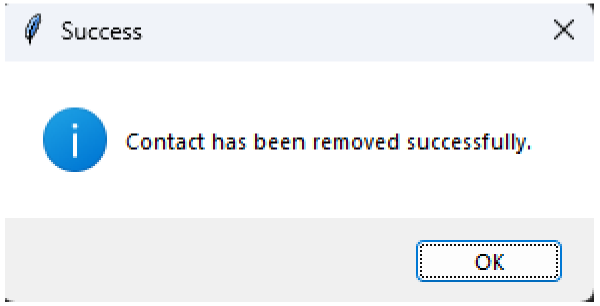
2.3 – Testing Information

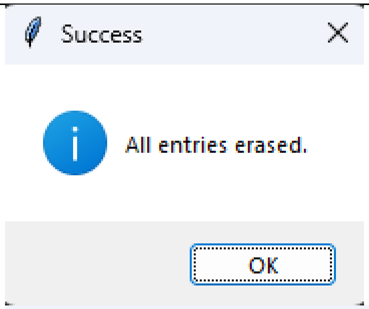
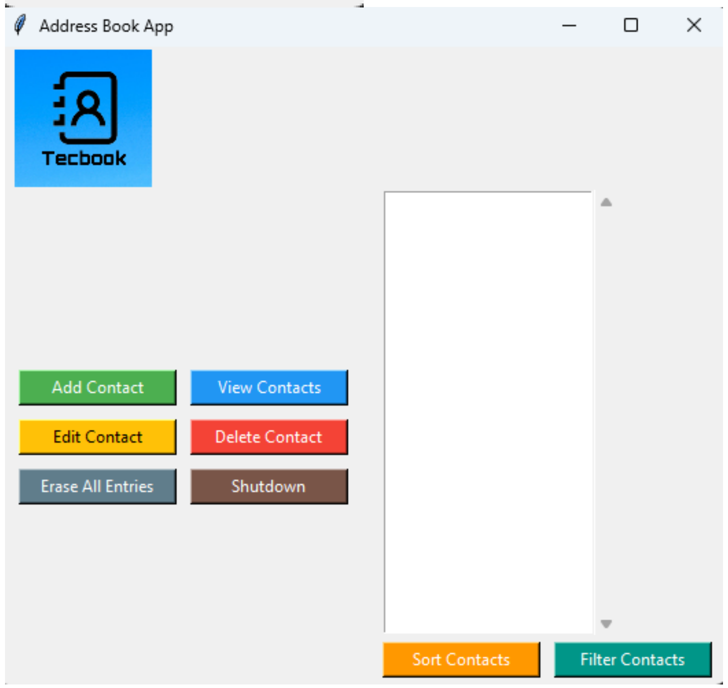
The members P2785659, P2724555, P2796362 and P2836714 manually tested the application address book. The testing encompassed features like sort/filter contacts, adding new contact, removing a contact, updating information, and seeing combined details. The table below summarizes all of the testing situations with their predicted outcomes and actual results.

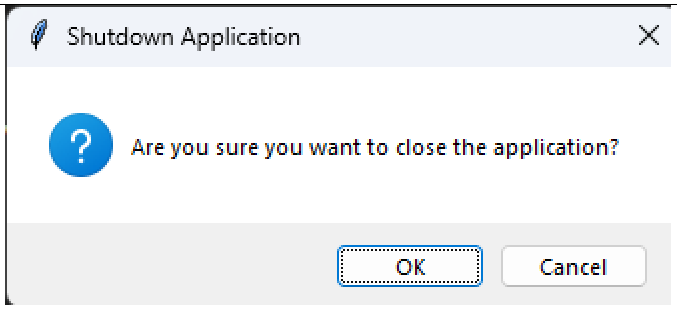
Test Case	**Feature/Functionality**	**Test Steps**	**Expected Outcome**	**Actual Outcome**	**Pass/Fail and screenshot**
TC-001	Application Launch	1. Execute **main.py**	Application window opens successfully	Application window opened successfully	Pass  Address Book App Tecbook Add Contact View Contacts Edit Contact Delete Contact Erase All Entries Shutdown Sort Contacts Filter Contacts
TC-002	Add Contact	1. Click "Add Contact" button 2. Enter contact details in the dialog 3. Click "OK"	New contact added to the address book	New contact added successfully	Pass  Address Book App Tecbook John Smith Add Contact View Contacts Edit Contact Delete Contact Erase All Entries Shutdown Sort Contacts Filter Contacts

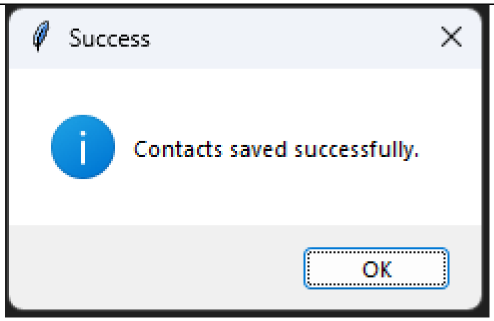
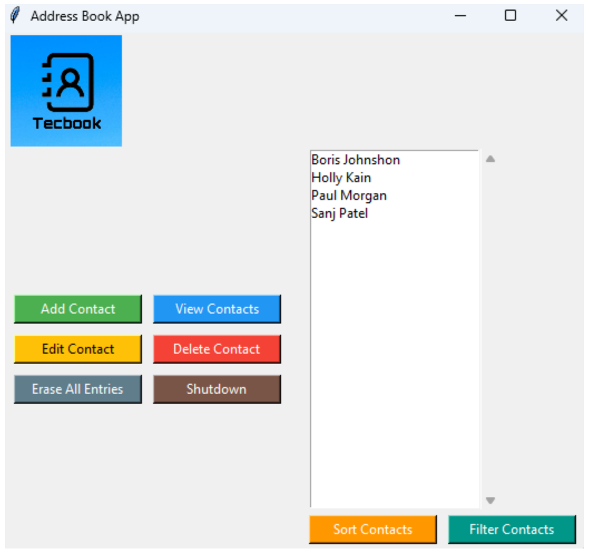
TC-003	View Contacts	1. Select the contact 2. Click "View Contacts" button	Selected contact should be displayed	Contact displayed as expected	Pass  John Smith's Details First Name: John Last Name: Smith Address: 12 Watkin Road Mobile Number: 0771983915 Secondary Number: 0782347812 Email Address: johnsmith@gmail.com
TC-004	Edit Contact	1. Select a contact 2. Click "Edit Contact" button 3. Modify contact details 4. Click "OK"	Contact details updated	Contact details updated successfully	Pass

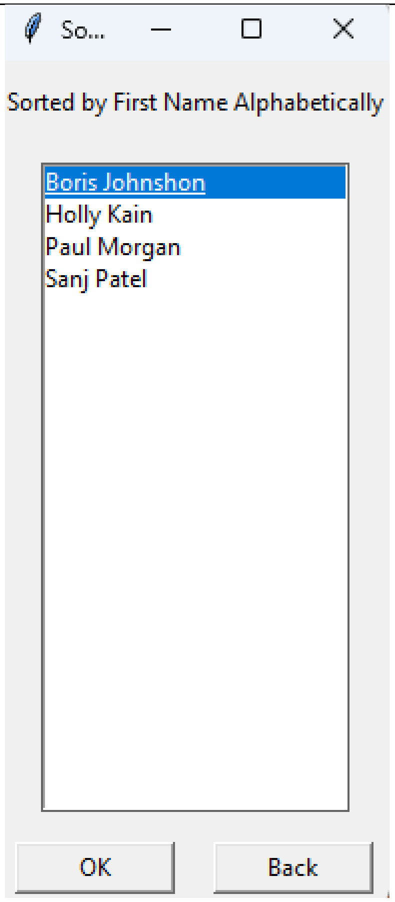
					 Edit Contact First Name: John Last Name: Smith Address: 12 Watkin Road Mobile Number: 0771983915 Secondary Number: 0782347812 Email Address: john@outlook.com Insert a photo: D:/Users/Shyam Dattani/Dov  Success Contact has been updated successfully.
TC-005	Delete Contact	1. Select a contact 2. Click "Delete Contact" button 3. Confirm deletion	Contact deleted from the address book	Contact deleted successfully	Pass

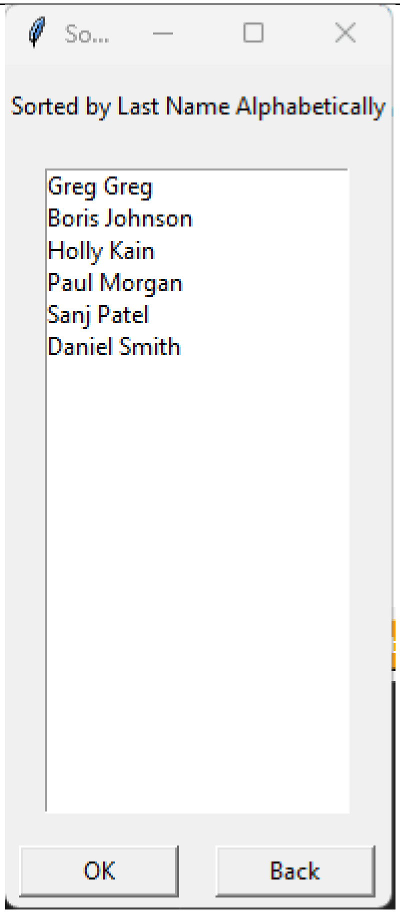
					 Delete Contact Are you sure you want to delete this contact? Yes No  Success Contact has been removed successfully. OK
TC-006	Erase All Entries	1. Click "Erase All Entries" button 2. Confirm erasure	All contacts removed from the address book	All contacts erased successfully	Pass

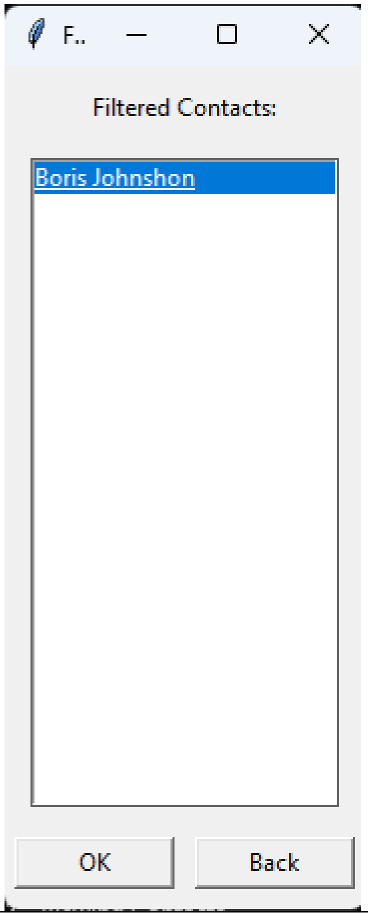
					 Success All entries erased. OK  Address Book App Tecbook Add Contact View Contacts Edit Contact Delete Contact Erase All Entries Shutdown Sort Contacts Filter Contacts
TC-007	Shutdown Application	1. Click "Shutdown" button 2. Confirm shutdown	Application closes without errors	Application closed successfully	Pass

					 Shutdown Application Are you sure you want to close the application? OK Cancel
TC-008	Save Contacts to File	1. Add multiple contacts 2. Click "Shutdown" button	Contacts saved to file	Contacts saved successfully	Pass

					 Success Contacts saved successfully.
TC-009	Load Contacts from File	1. Restart the application	Contacts loaded from file	Contacts loaded successfully	Pass  Address Book App Boris Johnshon Holly Kain Paul Morgan Sanj Patel Tecbook Add Contact View Contacts Edit Contact Delete Contact Erase All Entries Shutdown Sort Contacts Filter Contacts
TC-010	Sort Contacts (by First Name)	1. Click "Sort Contacts" button 2. Choose "First Name Alphabetically"	Contacts sorted alphabetically by first name	Contacts sorted successfully	Pass

					 So... Sorted by First Name Alphabetically Boris Johnshon Holly Kain Paul Morgan Sanj Patel
TC-011	Sort Contacts (by Last Name)	1. Click "Sort Contacts" button 2. Choose "Last Name Alphabetically"	Contacts sorted alphabetically by last name	Contacts sorted successfully	Pass

					 So... Sorted by Last Name Alphabetically Greg Greg Boris Johnson Holly Kain Paul Morgan Sanj Patel Daniel Smith
TC-012	Filter Contacts	1. Click "Filter Contacts" button 2. Enter filter conditions 3. Click "Apply Filter"	List of contacts filtered based on conditions	Contacts filtered successfully	Pass

					 Filtered Contacts: Boris Johnshon

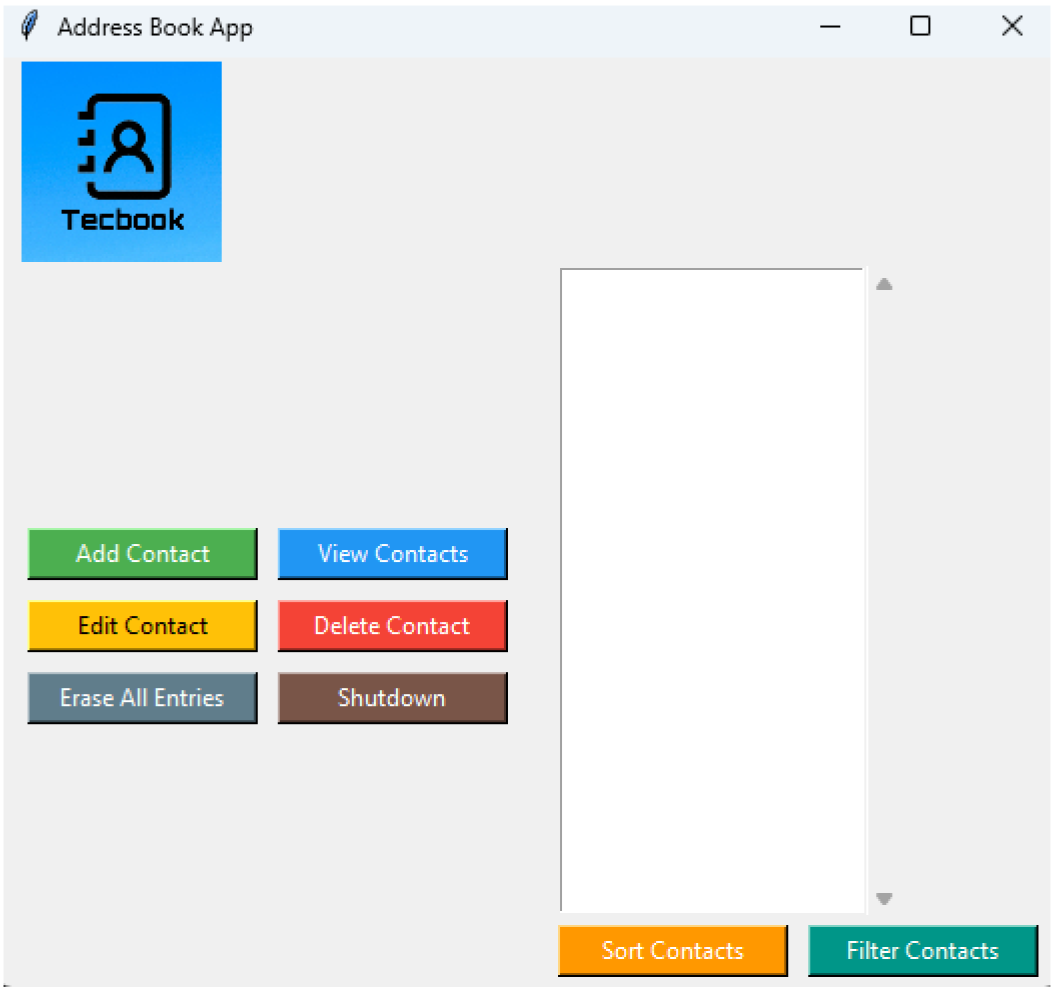
This part presents a detailed overview of documentation associated to the Address Book application: design choices selection, architecture that concentrates on usability and simplicity while simplifying source code for key classes based in an object-oriented philosophy. These findings of the testing underlined the necessity to perform comprehensive tests in numerous conditions to assure contact information storage stability and eventually a smooth user-interface.

3 – End User Guide

Welcome, Address Book App's user. The following supplied end user guide supplies heedful guidance to supplement from your delight at our application. The Address Book is basic yet the most helpful tool that organizes and synchronises your contacts. For each step, images will also be provided to aid the user grasp better.

3.1 – Getting Started

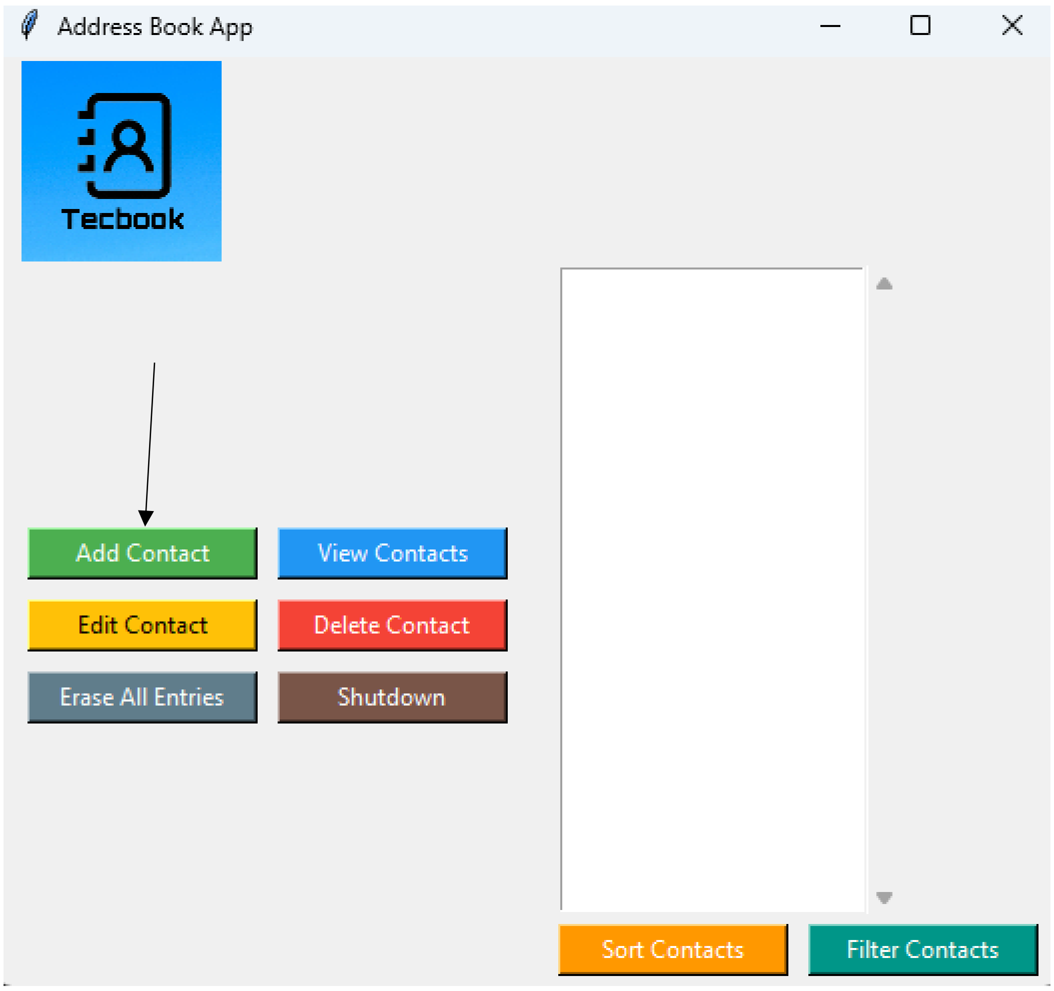
After initialising the Address Book App, an interface is provided with several buttons for contact management whereby once the button is pressed, the action relevant to the button will be executed.



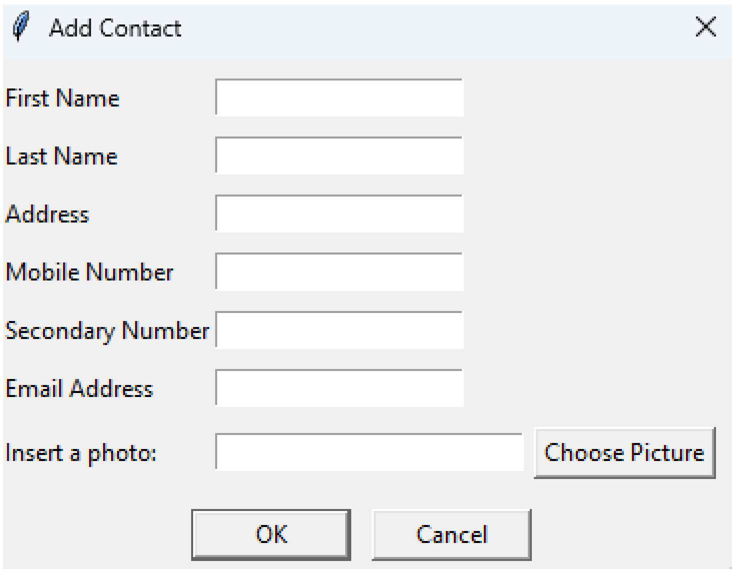
3.2 – Adding a new Contact

For adding a contact:

Press the "Add Contact" button.

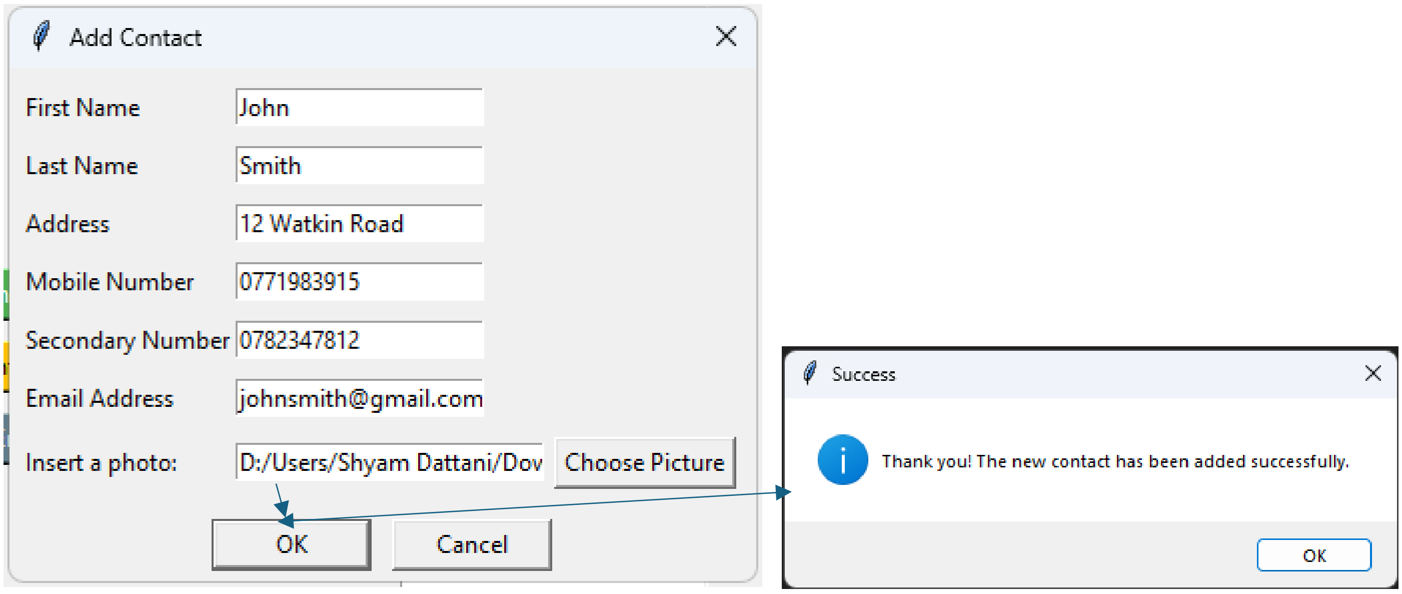


A dialog box will popup, add the new contact data such as First Name, Last name, Address, Mobile Number, Secondary Number, Email address and inserting a photo.


Add Contact

First Name	
Last Name	
Address	
Mobile Number	
Secondary Number	
Email Address	
Insert a photo:	

Save the new contact with the OK button and a confirmation message will appear stating that the contact has been saved.


Add Contact

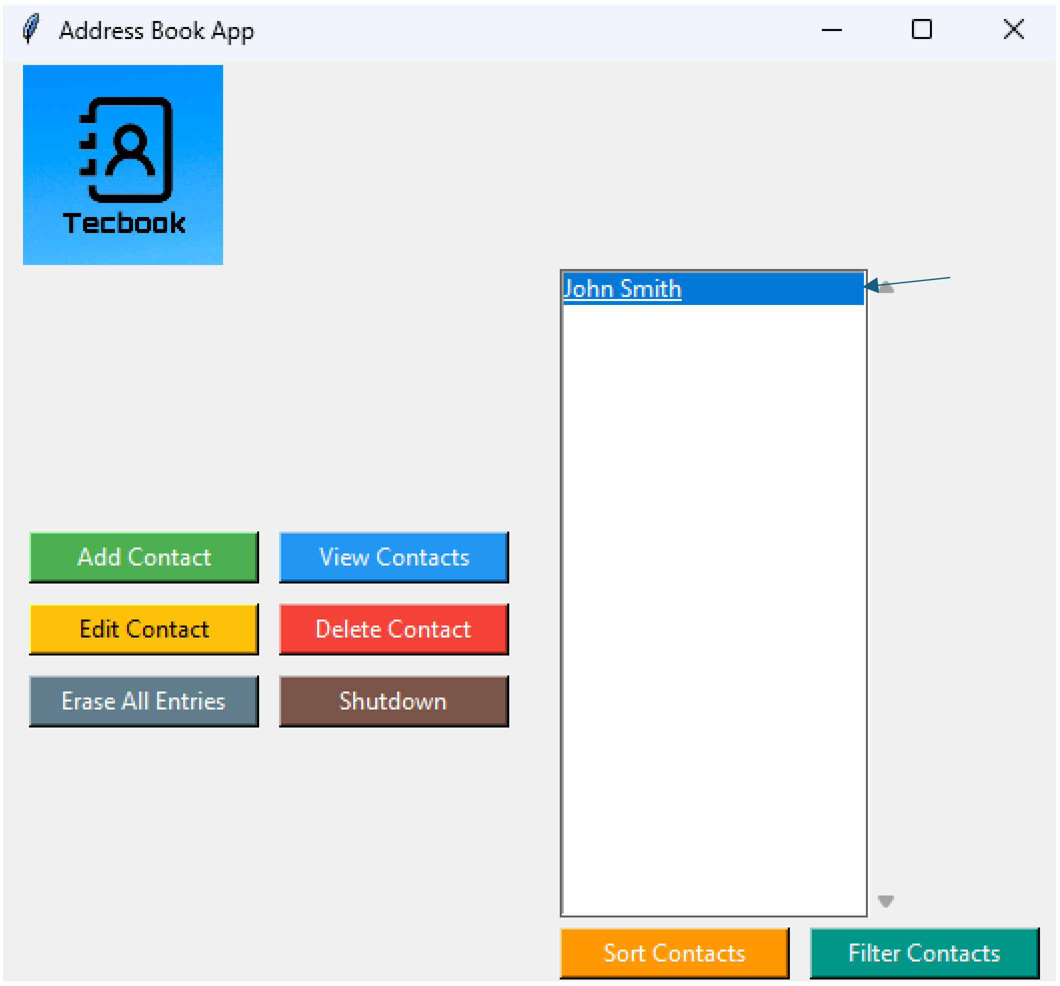
First Name	John
Last Name	Smith
Address	12 Watkin Road
Mobile Number	0771983915
Secondary Number	0782347812
Email Address	johnsmith@gmail.com
Insert a photo:	D:/Users/Shyam Dattani/Dov

Thank you! The new contact has been added successfully.

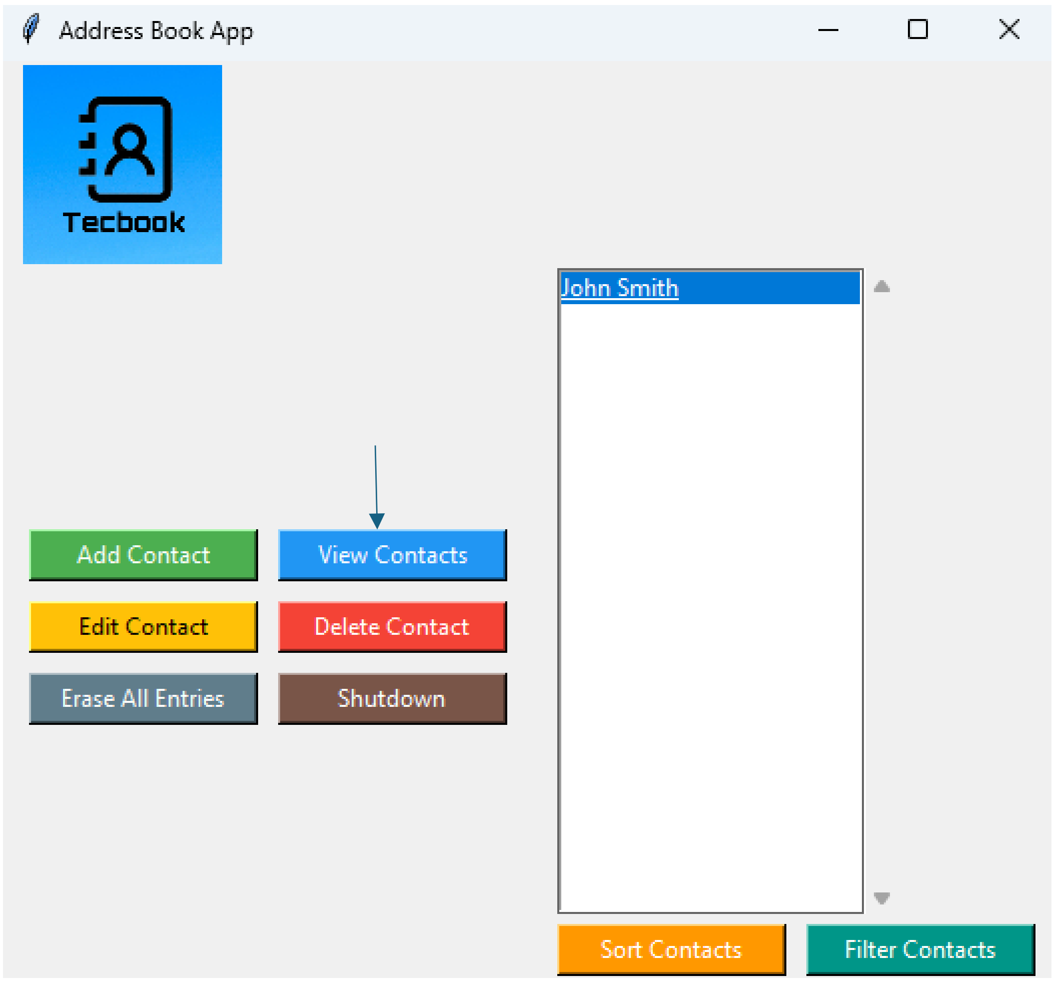
3.3 – Viewing Contacts

To obtain contact information:

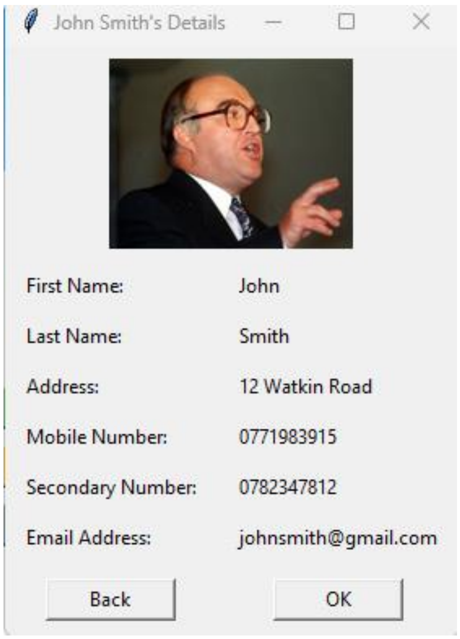
Select the required contact out of the list.



There is "View Contact" button surrounding each contact which you are interested in.



A click on one of the offered contacts will reveal the contact's entire name, the specified contact's address, phone number, secondary number and email address in another window. It will reveal the connected image of protrusion on that window if available.

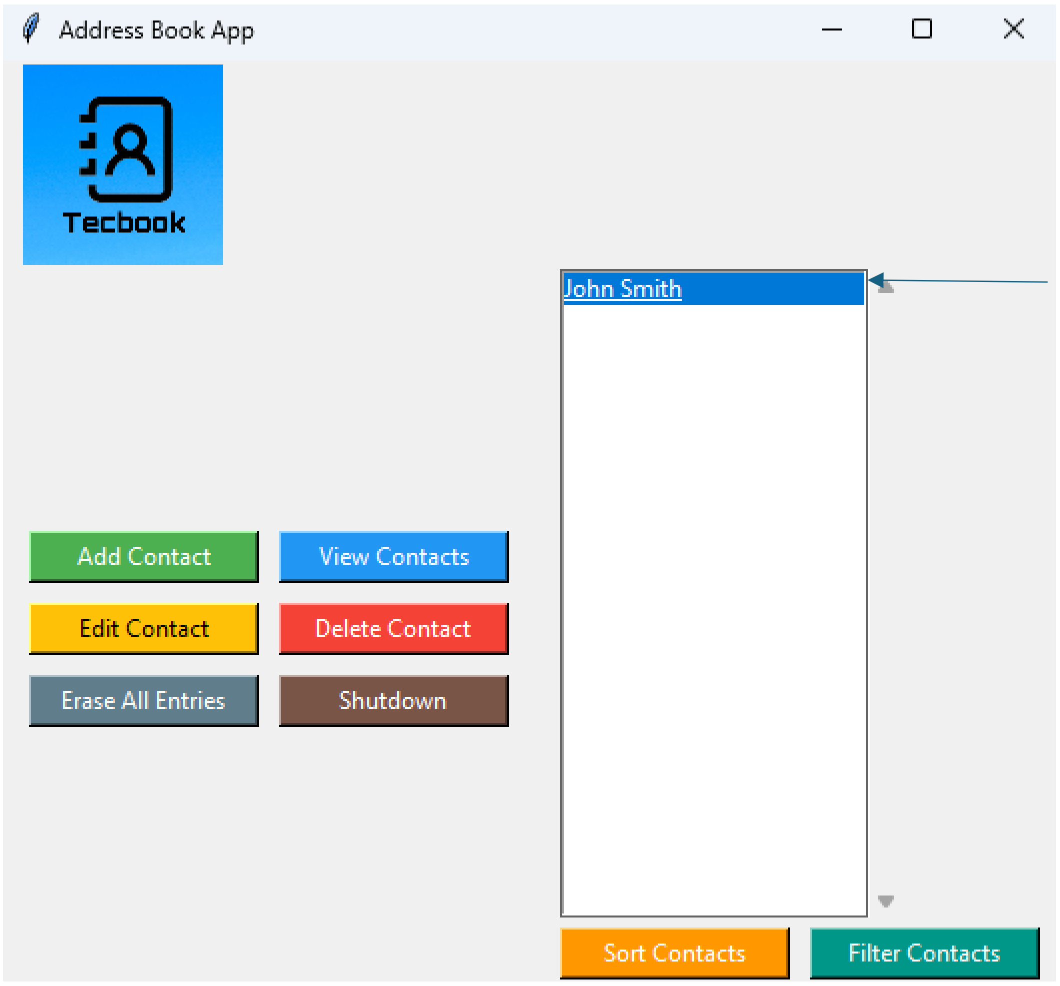

John Smith's Details

First Name:	John
Last Name:	Smith
Address:	12 Watkin Road
Mobile Number:	0771983915
Secondary Number:	0782347812
Email Address:	johnsmith@gmail.com

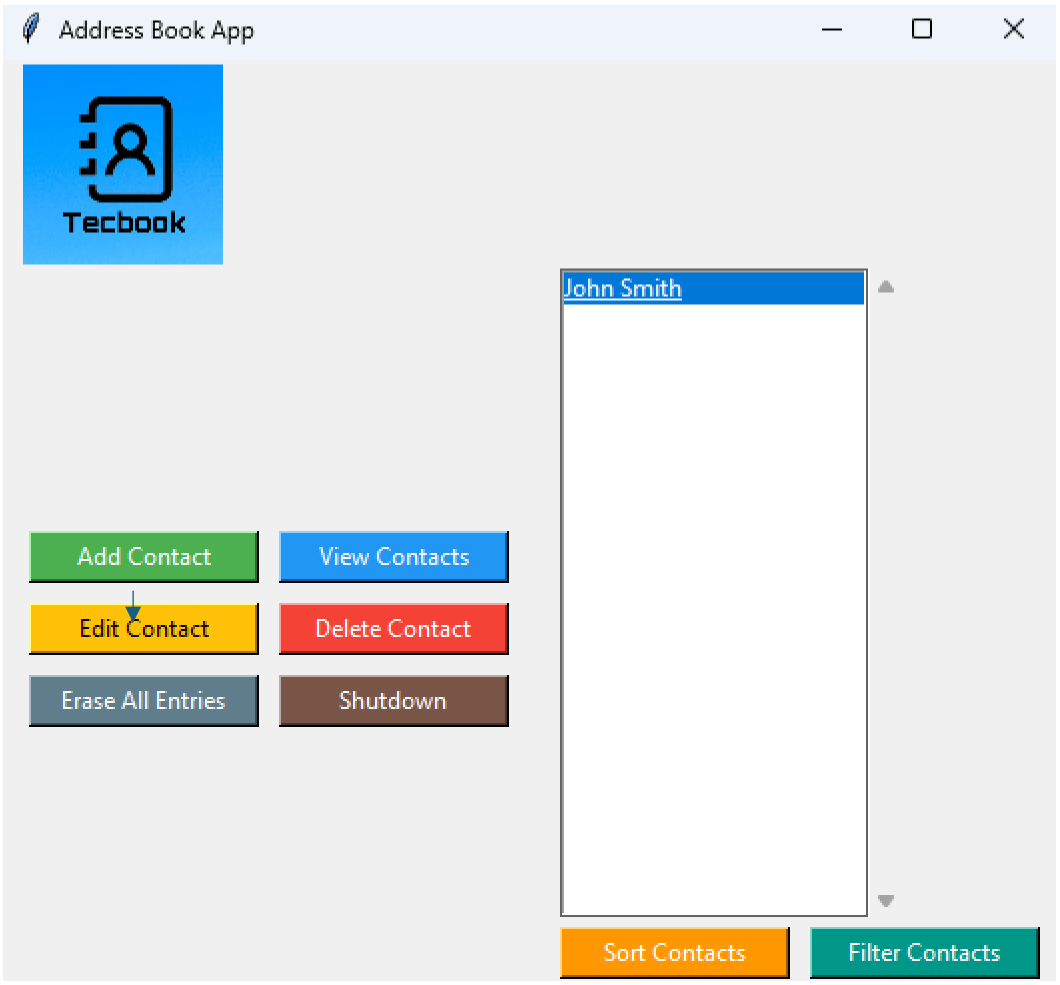
3.4 – Editing Contacts

To edit a contact:

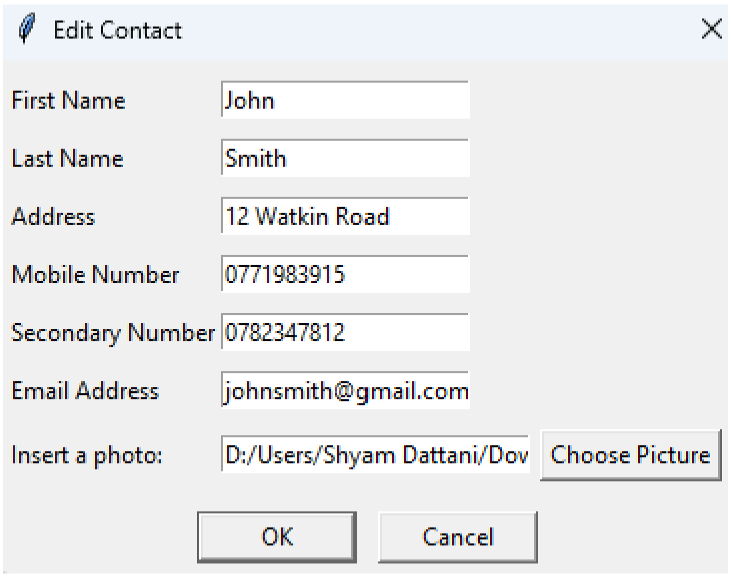
Choose the contact to be modified from list.



Click on "Edit Contact" button.

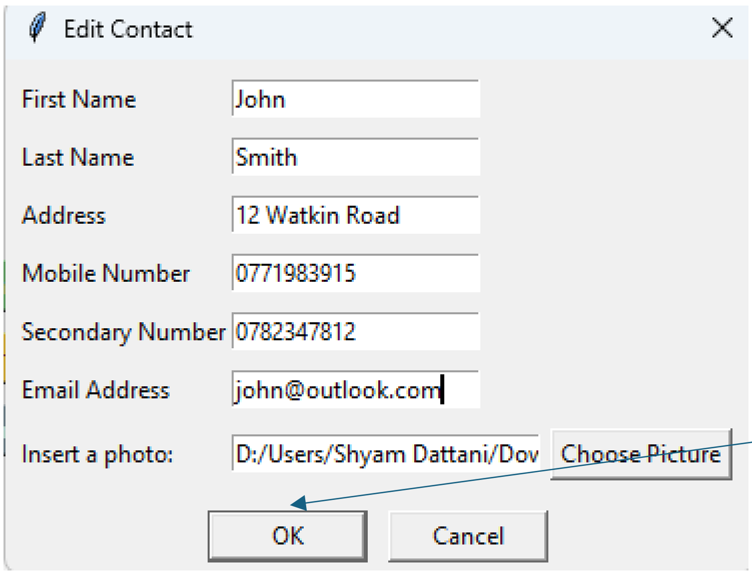


The supplied contact info will come up which is in a box.

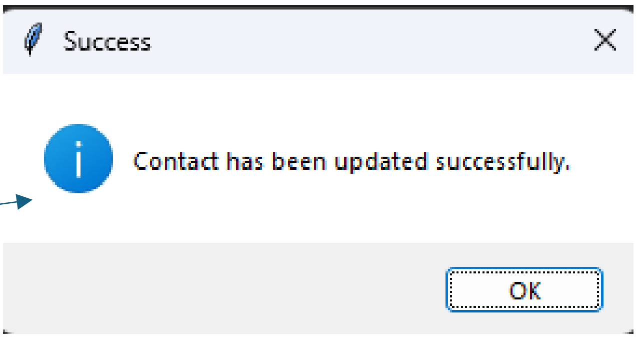


First Name	John
Last Name	Smith
Address	12 Watkin Road
Mobile Number	0771983915
Secondary Number	0782347812
Email Address	johnsmith@gmail.com
Insert a photo:	D:/Users/Shyam Dattani/Dov

Edit to the required information and hit 'OK' to apply changes and a confirmation message will appear.


Edit Contact

First Name	John
Last Name	Smith
Address	12 Watkin Road
Mobile Number	0771983915
Secondary Number	0782347812
Email Address	john@outlook.com
Insert a photo:	D:/Users/Shyam Dattani/Dov

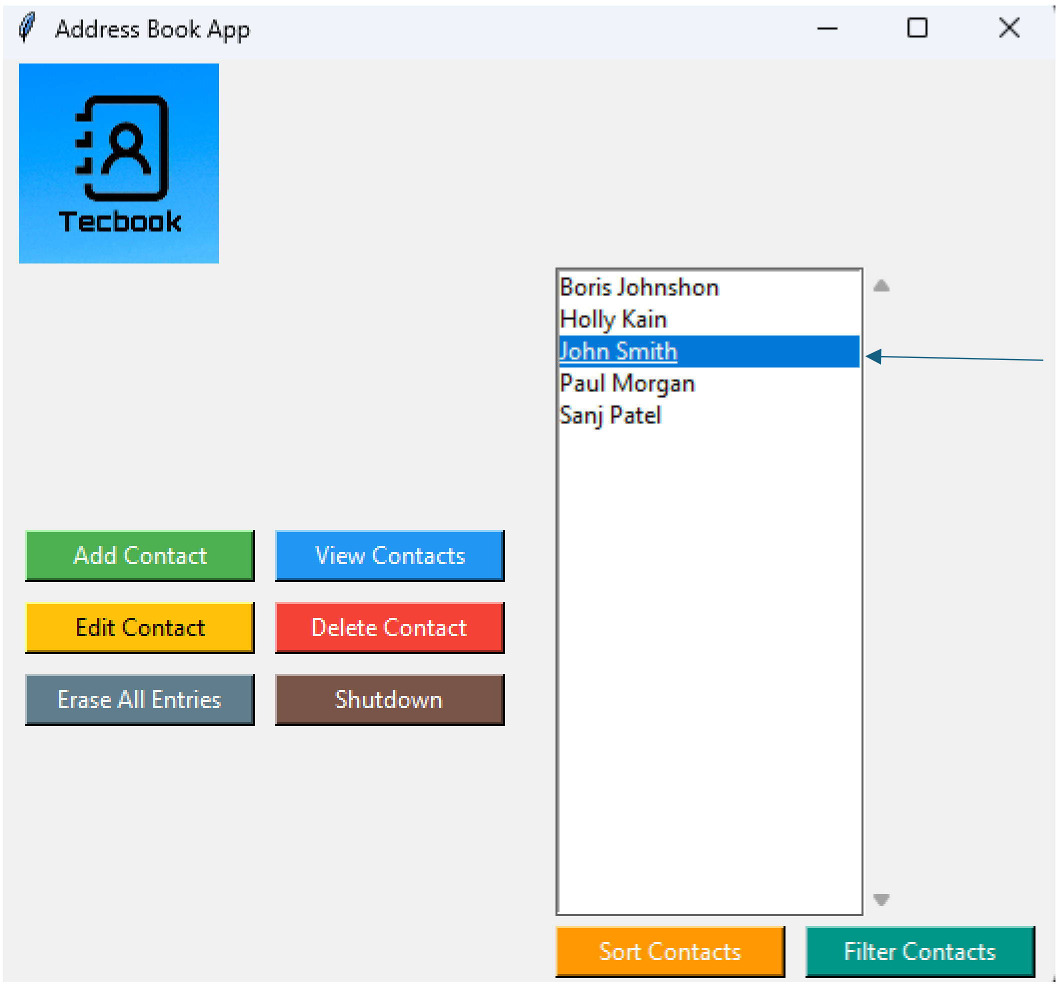

Success

Contact has been updated successfully.

3.5 – Deleting Contacts

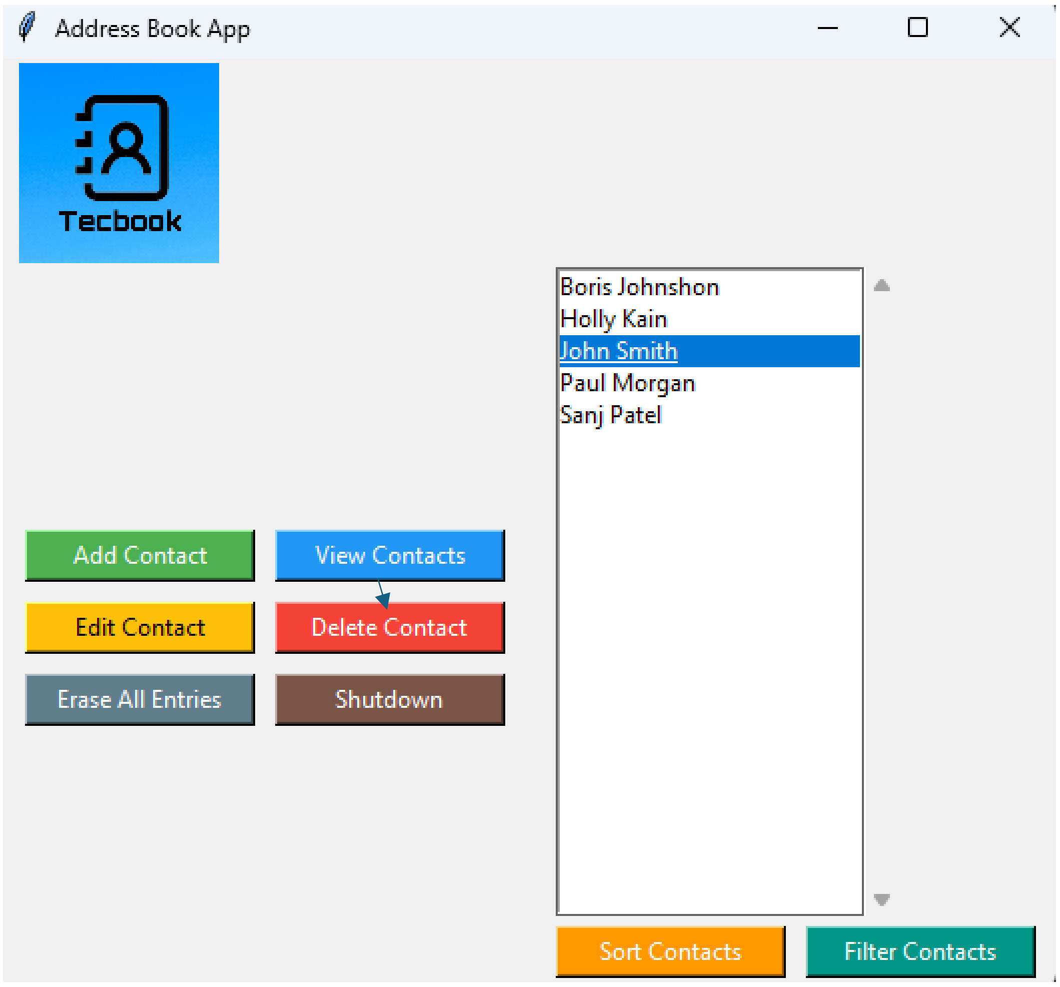
To delete contact:

Choose the contact to remove by selecting the contact.

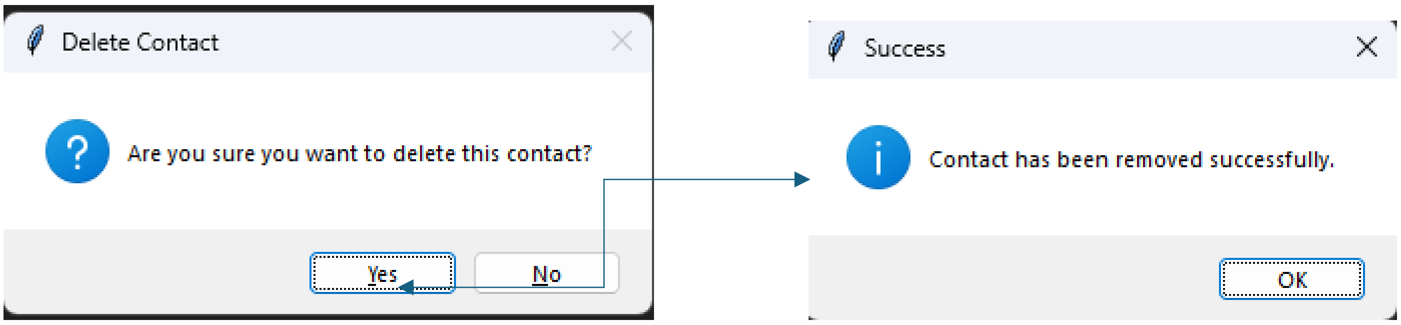

Address Book App

Boris Johnshon
Holly Kain
John Smith
Paul Morgan
Sanj Patel

Once you have selected the contact that you wish to remove, click on ‘Delete Contact' option.



If genuinely wish to erase the contact, accept from prompt and once you’ve accepted the prompt a confirmation message will appear.

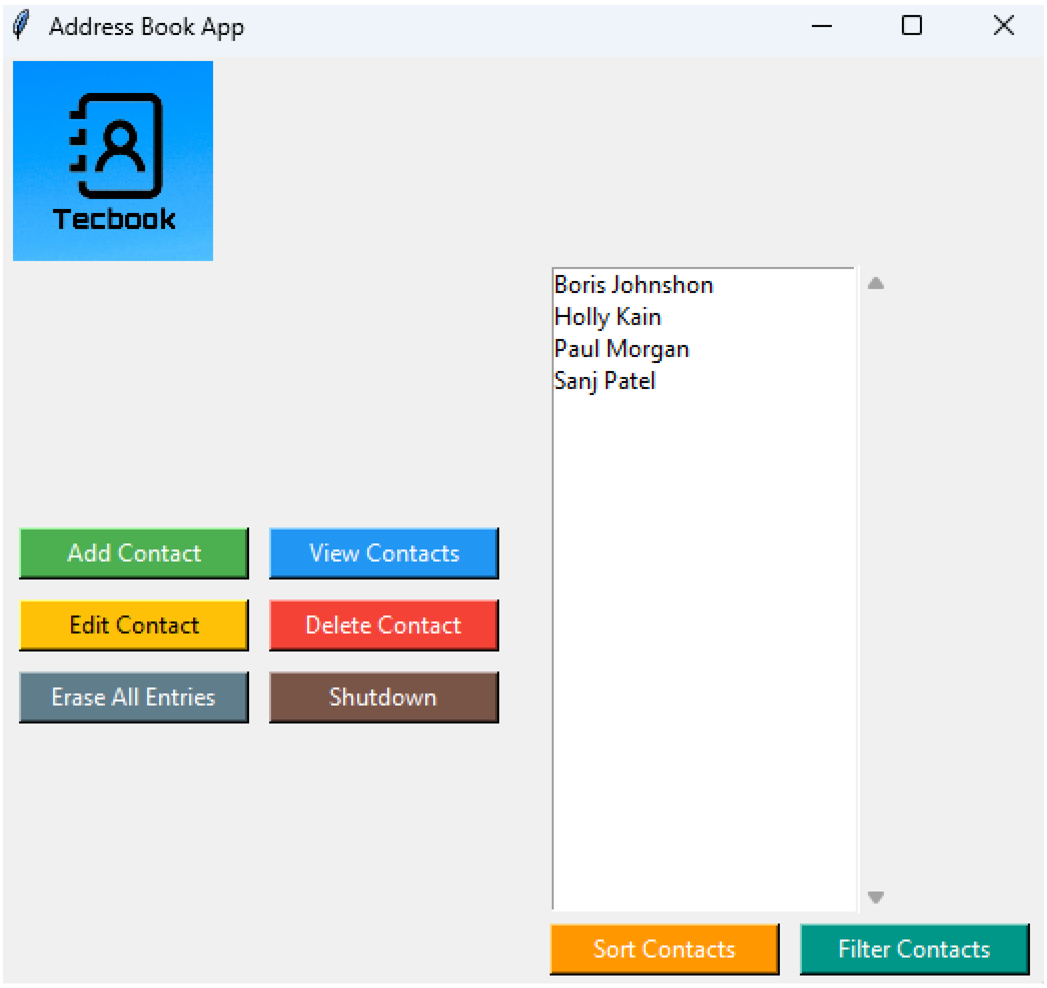

Delete Contact

Are you sure you want to delete this contact?

Success

Contact has been removed successfully.

Now the address book will be updated with chosen the contact removed.

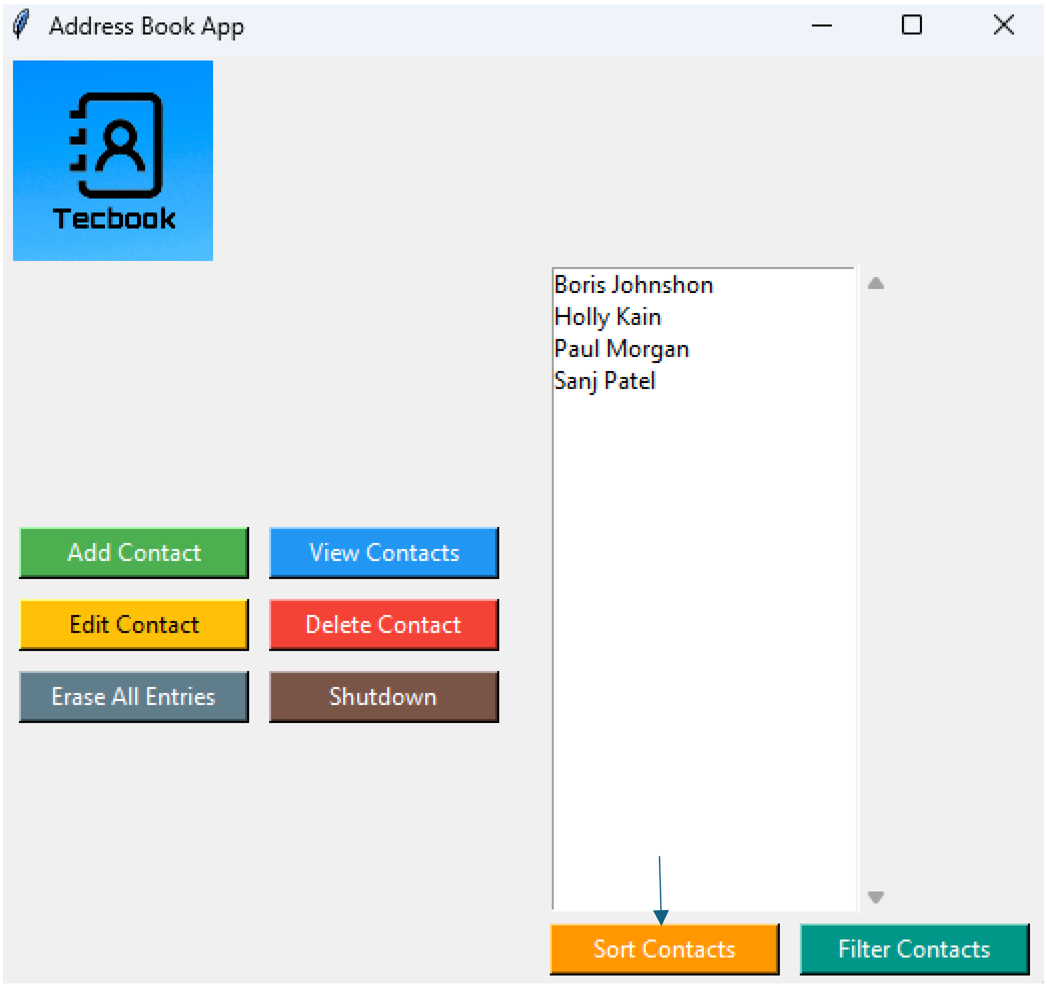


Boris Johnshon
Holly Kain
Paul Morgan
Sanj Patel

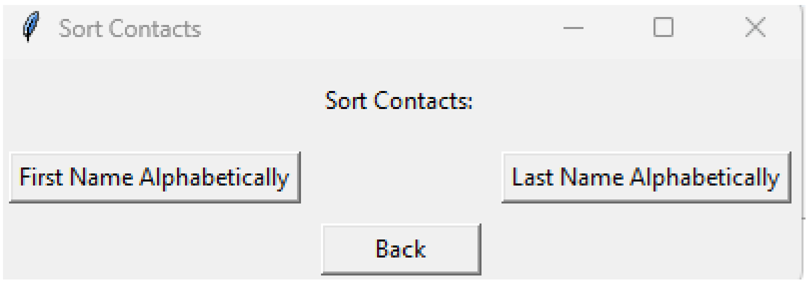
3.6 – Sorting Contacts

Contacts sorting:

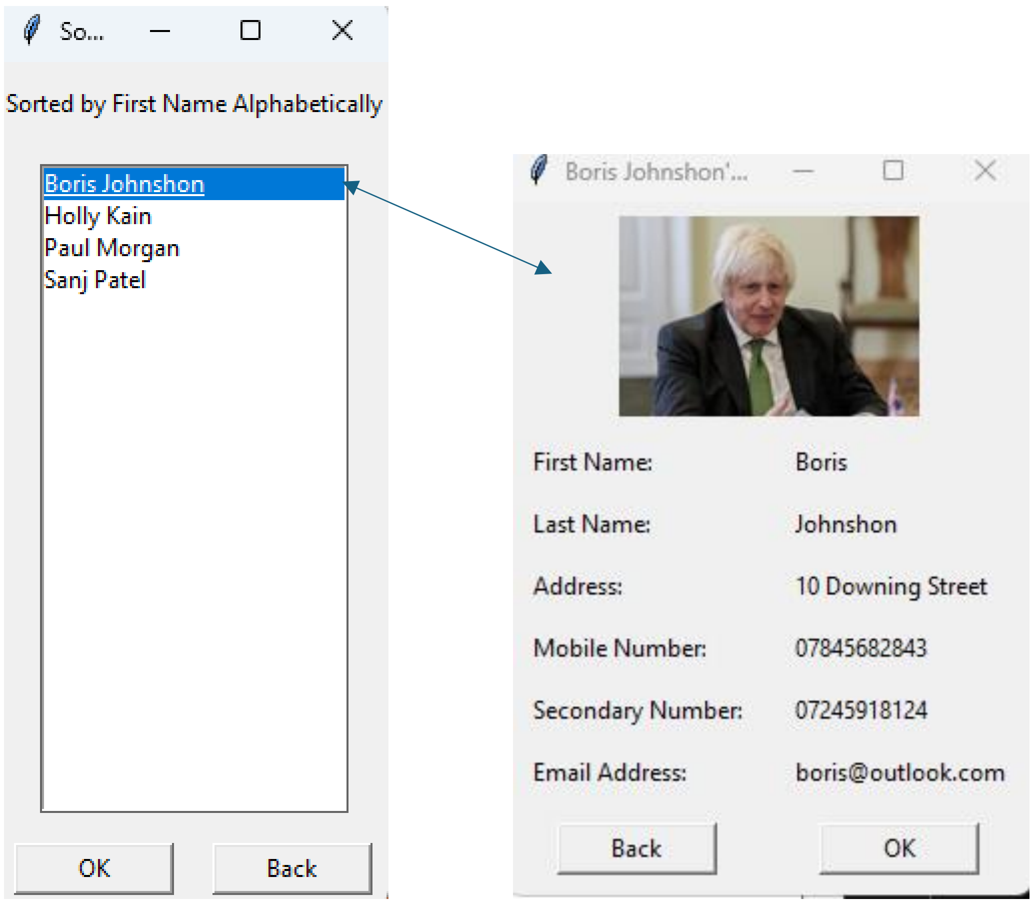
Click on "Sort Contacts" button.



From the list of various sorting algorithms, pick between Alphabetical sorting by First Name or Last Name.



Selected order is instantly transmitted to contacts, and you can double click on the contact to view the contact details of that specific contact.



Sorted by First Name Alphabetically

Boris Johnshon
Holly Kain
Paul Morgan
Sanj Patel

OK Back

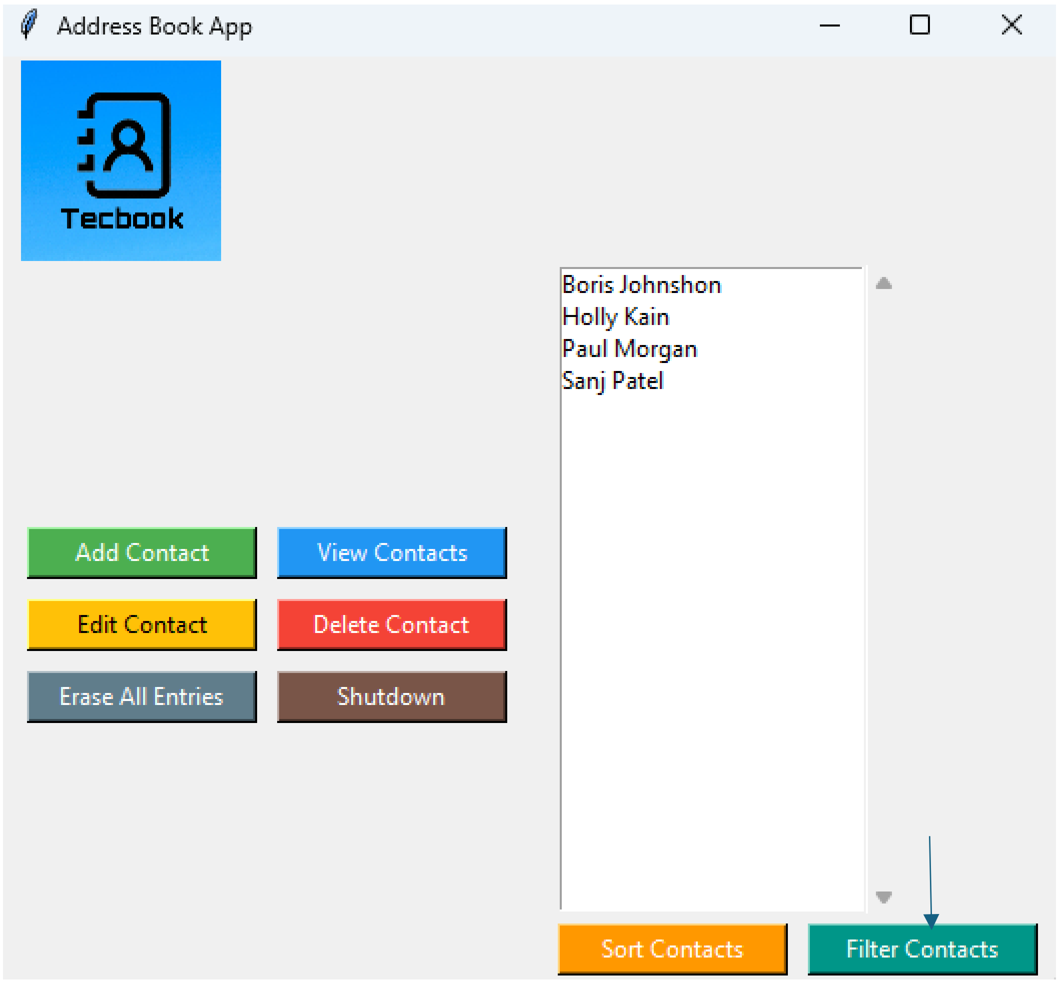
First Name:	Boris
Last Name:	Johnshon
Address:	10 Downing Street
Mobile Number:	07845682843
Secondary Number:	07245918124
Email Address:	boris@outlook.com

Back OK

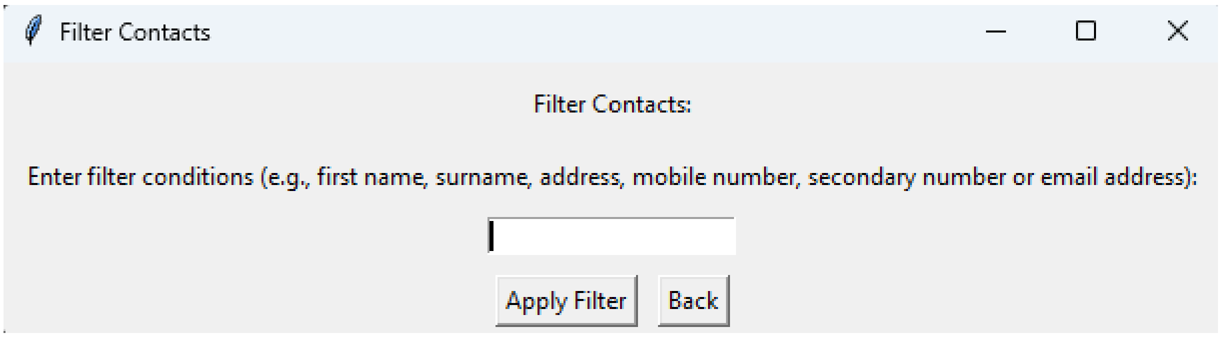
3.7 – Filtering Contacts

Filtering contacts:

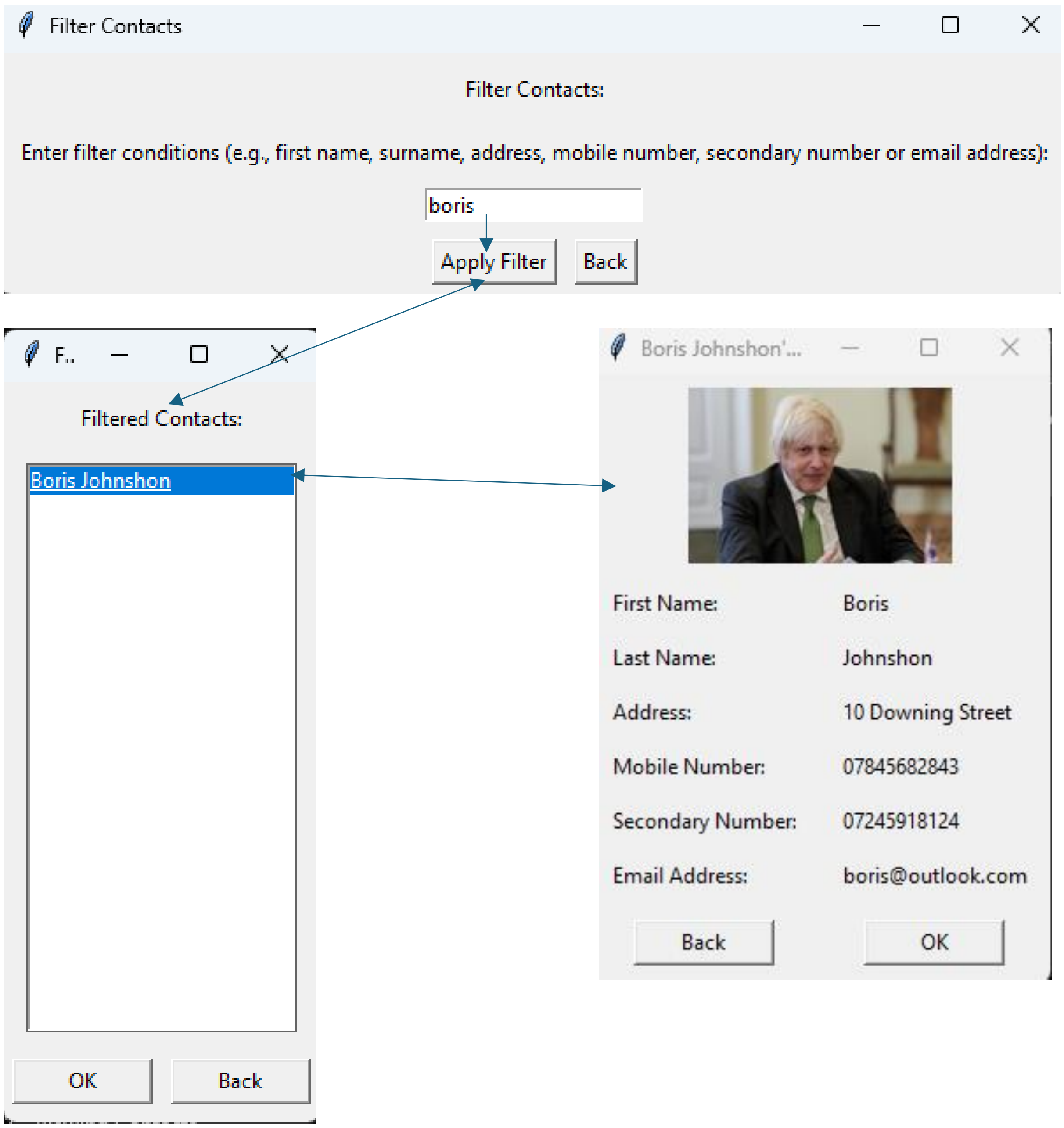
Click on click "Filter Contacts".



Specify filter criteria using first name, last name, address, mobile number, secondary number or email address.


Filter Contacts
Filter Contacts:
Enter filter conditions (e.g., first name, surname, address, mobile number, secondary number or email address):
Apply Filter
Back

Press "Apply Filter" in order to display contacts meet in the criteria and you can also double click on a contact to view the contact details.



Filter Contacts

Filter Contacts:

Enter filter conditions (e.g., first name, surname, address, mobile number, secondary number or email address):

boris

Apply Filter Back

F..

Filtered Contacts:

Boris Johnshon

OK Back

Boris Johnshon'...

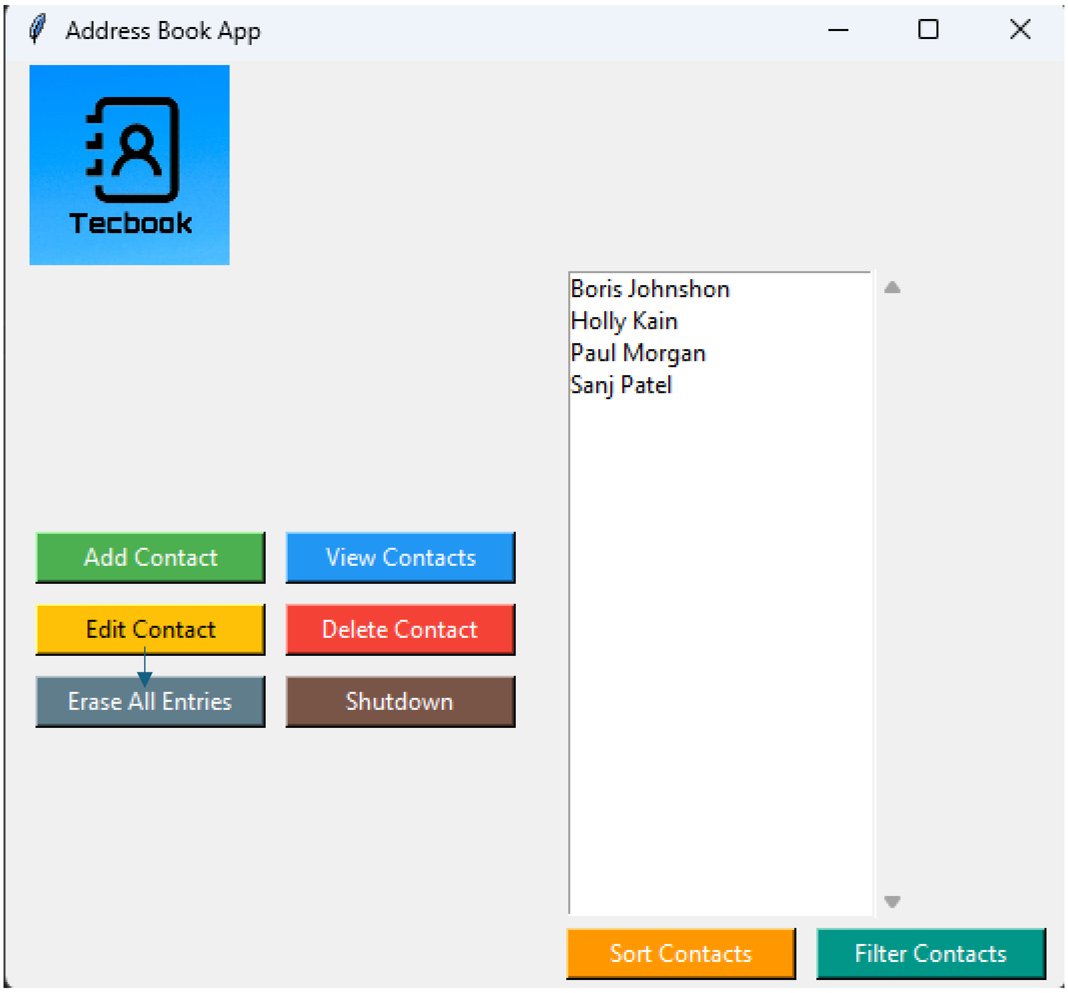
First Name:	Boris
Last Name:	Johnshon
Address:	10 Downing Street
Mobile Number:	07845682843
Secondary Number:	07245918124
Email Address:	boris@outlook.com

Back OK

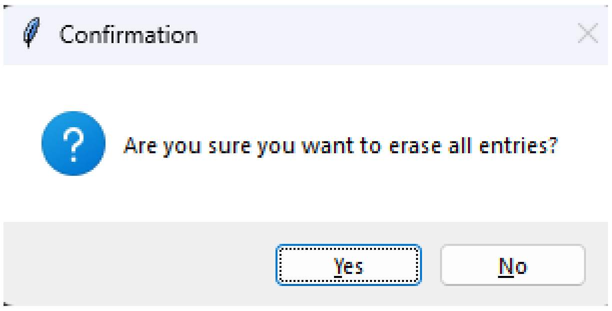
3.8 – Erasing All Entries

To remove all entries:

Please press "Erase All Entries" button.

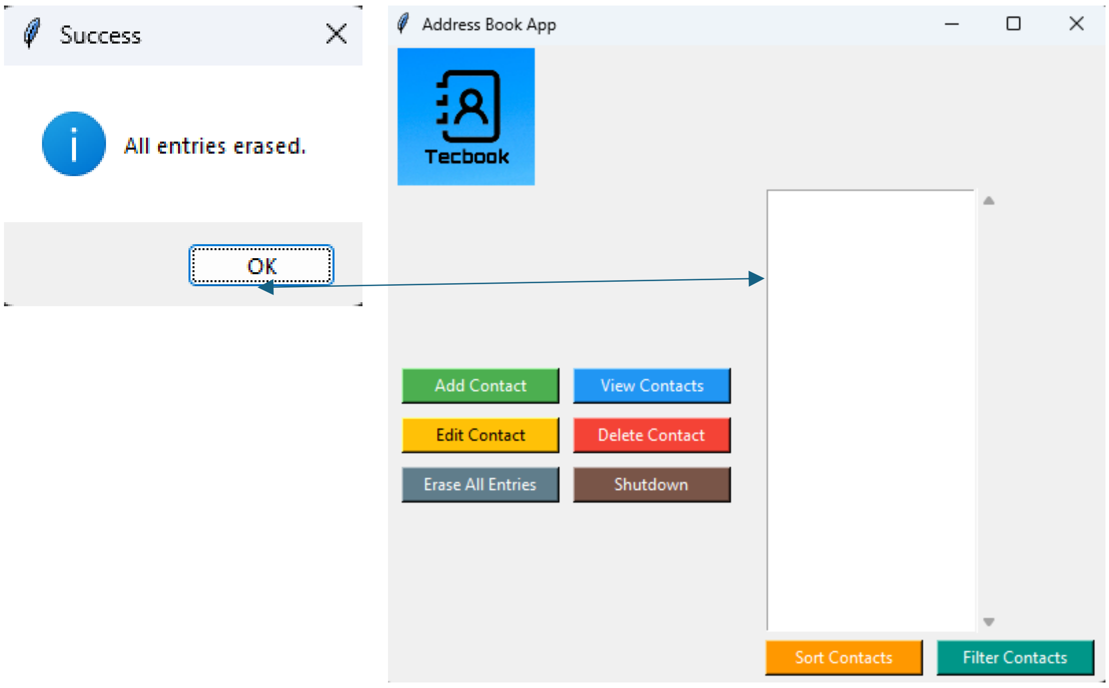


Confirm if requested for erasing.



Are you sure you want to erase all entries?

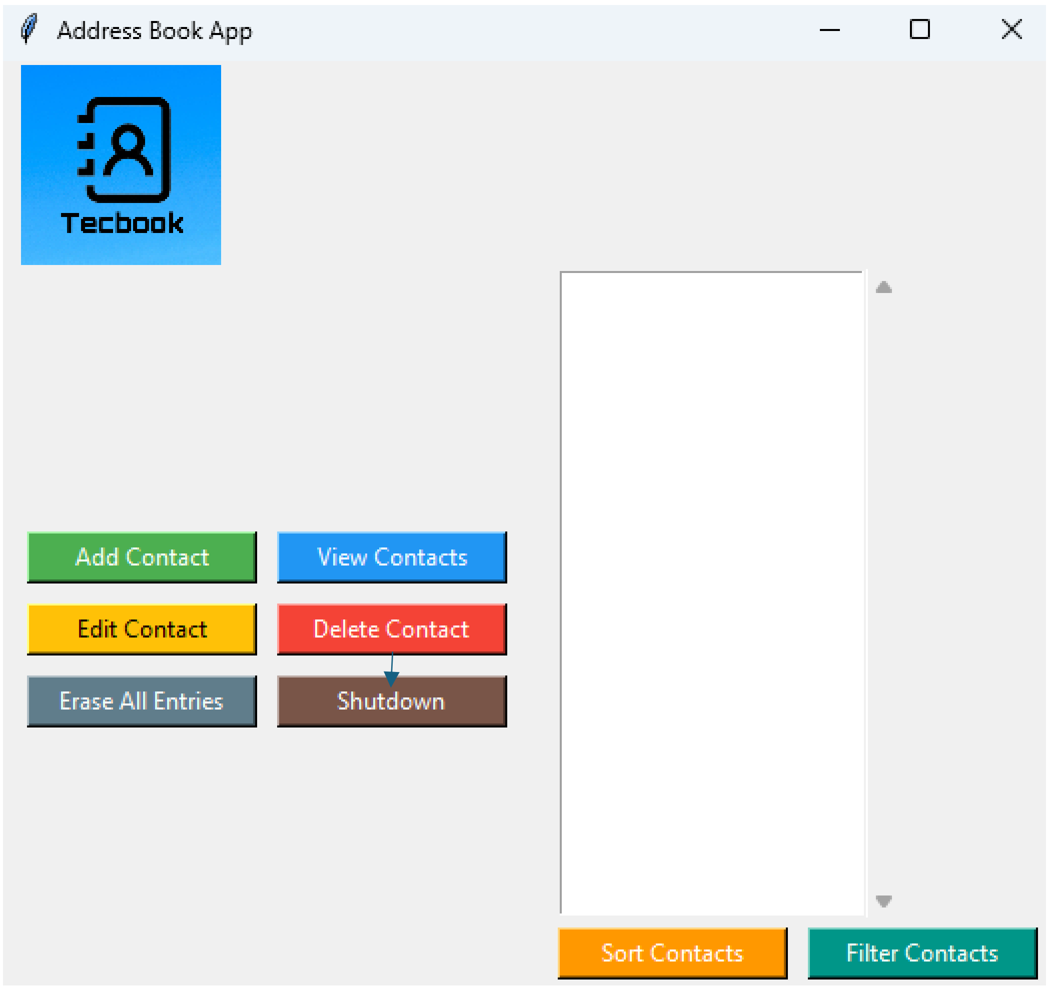
All contacts will be wiped and confirmation will appear.


All entries erased.

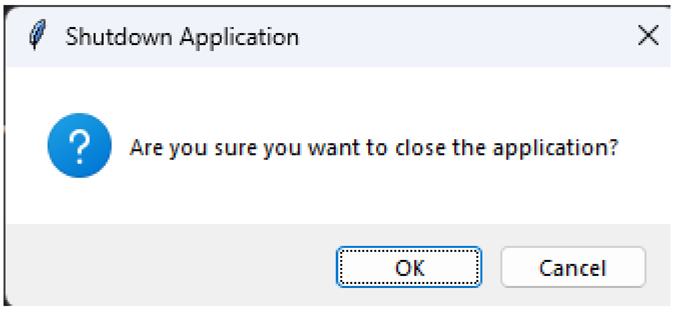
3.9 – Shutting Down the Application

The programme may be shut off by:

Press the "Shutdown" button.

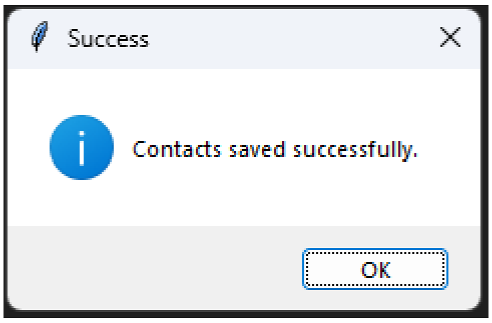


Confirming the action when requested for the same.



Are you sure you want to close the application?

The programme is terminated, and the contacts are stored in it.


Success

Contacts saved successfully.

This part includes all the user instructions for dealing with the Address Book application.

Users may swiftly add any view, in addition to editing and removing contacts. Contacts Management additionally enables the user to rapidly manage sort and filter choices as well but their simplicity is centred upon how the users may actually effectively use Address Book services.

4 - Usability/User Experience & Design Enhancements

This part would mainly be for the many modifications and improvements done to the overall usability, user experience as well as design of the address book programme performed. From the quick contact entering interface to sorting and filter selections, colourful graphics displays every component capturing users attention. This is a desire for this experience on our product as smooth sailings in its end-users. They include the user-friendly layout and the availability of such a unique brand with design components carefully developed for pleasant use and outstanding interface efficiency are some of the features.

4.1 – User-Friendly Contact Entry Dialog

With this, the input dialog developed for contact. Inputting of contact information is credited to simple usage of an intuitive interface set out in multiple categories such as First Name, Last Name, Address, Mobile Number, Secondary Number, Email Adress and Upload Image.

4.2 – Clear Picture Display

Viewing or editing of a contact, in the newest version of this programme when photo search discovered clear and desired view. The presentation of the photograph is brought forward with suitable thumbnail size thereby assuring attractiveness of the display.

4.3 – Improved Handling of the Picture

The “Insert Photo” is included as provision to convenience in picking the contact photographs. This capability is provided to the input box where users are able to examine the picked picture.

4.4 – Advanced Contact Editing

Through superior effectiveness contact editing, it is intended at making work easier and reduces several phases. Clear notifications appear in the very application when precisely altered contacts wherein a user may modify on specifics of one contact simple.

4.5 – Sort and Filter Functionality

The programme has sorting option and filtering options. supplies a way of finding and with regards to sort out any contact in accordance with their names which can just simply be sorted either in First Name or Last Name depending on the alphabet. And filtering of the contacts may also use a free form text field, as well as with the use of various search factors, e.g., the first name, last name, address, mobile number, secondary number or email.

4.6 – Visual Elements and Logo

The application logo is astonishingly interesting which may be regarded as professional and corporate-centred. There is also an area for the installation of a logo, sits left side at the top.

4.7 – Confirmation Messages

Confirmation messages have also been modified ensuring polite and user helpful comments are offered. For instance, sending relevant success notification following successful addition or change and as well as deletion of contacts.

4.8 – Error Handling

The update now addresses error handling with regard to the contact info inputs. Even more will be the cautions that consumers get in case of troubles such as when storing and loading contacts established using this applications contact-taking application, delivering a more sophisticated form whose service does not fail so quickly.

4.9 – Intuitive Buttons

For the layout, buttons to add contacts in various colours and clear text labels in most aesthetic form are complementing all these structures as a whole. This would be advantageous for an end user since user may define his/her tasks exactly and carry them swiftly.

4.10 – Overall Layout and Structure

The layout arrangement demonstrates that it is a classy designed streamlined clean application. Basically, it has a straightforward flow generally when one scrolls over the buttons which have been arranged effectively.

In summary, these upgrades make the address book application more ergonomic and appealing as well as useful in general, which increases the usability of your product.

5 – Critical Evaluation

In this section, therefore, we shall explain the critical evaluation of the application address book that covers aspects on the experience of implementation and collaborative working, lessons learnt from module feedback, discussing the limitation of the application while suggesting ways to overcome these limitations for future enhancements.

5.1 – Experience of Deployment of the Application and Collaborative Working:

The experience is produced using the programme, shining largely at GUI creation using tkinter library and providing the required features. The strategy has embraced inputs from numerous diversified backgrounds which has gone on to help towards enhance the overall quality of the application. Therefore, collaborating with the individuals required in correct moments have helped significantly in the workflow - discussing ideas as well as difficulties together. This clearly highlights why communicating and working together is so crucial in software development.

5.2 – Lessons Learned from Module Feedback:

While in presentation with the module, this input becomes crucial in the development of the application wherever within the team, the feedback has been on a note of appreciation for the design generally and the layout of the user interface which is comfortable to use. This feedback supported integrating such concepts in this development phase to make user interface substantially more intuitive. Adaptations were made to code exposed to requirements for maintaining closer to the best practices to contribute for higher readability of the code and boosting overall maintainability. Further, the insights concerning error handling and file operation did go a long way to making the programme a less unstable one. As a matter of fact, the experience from an iterative feedback loop has further highlighted how enhancing exception was a critical component to the complete process.

5.3 – Limitations and Future Enhancements

The software executes brilliantly its core duties, however there are a few things that it would be highly disruptive if not taken care of. For instance, during contact entry, input validation check has not been done and so it's feasible that the data supplied might be partial or even inaccurate. Robust input validation methods will be important for assuring the reliability and completeness of information given by the users.

One further significant addition may be accomplished by implementing a 'search' capability in the present programme. With the amount of contacts being added on, this would truly contribute to the user experience where any contact can be identified within seconds at most by typing few data alone.

Chime or ringtone when fields of requisites completed by the user in the application will create an air of liveliness and dynamic personality to build additional participation of the users. This not only increases the users' engagement with the program but also provides a delightful feel to the user interaction such that it makes the application seem more vibrant and living.

With reference to its visual representation, the GUI of the software is built naturally offering a visually basic exposure. Envisioning future iterations, however if the existing design choices were centred on a contemporary design element, there would be more perspective of the same examined. A bigger inclination superior looks and the user experience are what might improve the program's attractiveness.

Besides, if the programme is built to be dynamically responsive to varied sizes of displays and resolutions, it would make the application flexible for further future adjustments. The functionality would help a user work on the programme with ease at multiple devices and displays.

In such manner, such constraints may be acknowledged, where actively seeking customer input in the course of the development process functions as a source refining the product toward boosting customer happiness. A feedback loop has been important to our accomplishments in building a basis for future projects and guaranteeing a continual learning culture. While we pilot through these modifications, we hope that the application will guarantee that according to evolving demands and expectations of users it will be highly-polished and extremely simple to use.

6 – Conclusion

This conclusion of the study is all about the design and execution along with a vivid user experience using the Address Book application. Consequently, the records speak about system design, source code structure, testing procedures and user support that is related with the Address Book. Usability upgrades concentrate on a more user-friendly contact entry interface, better sorting and filtering, clearer image display. Critical evaluation comprised of experience of deployment, learning from module comments, constraints identified and anticipated future improvements. Joint efforts put in place towards coming up with the program successfully have supplied a polished, user-friendly and efficient address book application of which current progress may be made as well as future initiatives looked to.

7 – References

Amos, David. “Python GUI Programming with Tkinter – Real Python.” *Realpython.com*, 30 Mar. 2022, realpython.com/python-gui-tkinter/.

Babitski, Yan. “Getting Error Handling Right.” *The Startup*, 1 Jan. 2020, medium.com/swlh/getting-error-handling-right-9a1d39da0fa3.

Chu, Chih-Hsing, and Erh-Ting Kao. “A Comparative Study of Design Evaluation with Virtual Prototypes versus a Physical Product.” *Applied Sciences*, vol. 10, no. 14, 9 July 2020, p. 4723, https://doi.org/10.3390/app10144723.

Chris, K. (2023). *Visual Studio vs Visual Studio Code – What’s The Difference Between These IDE Code Editors?* [online] freeCodeCamp.org. Available at: https://www.freecodecamp.org/news/visual-studio-vs-visual-studio-code/.

Microsoft (2019). *Visual Studio*. [online] Visual Studio. Available at: https://visualstudio.microsoft.com/.

Reddy, S. (2022). *10 USEFUL VSCODE EXTENSIONS*. [online] Medium. Available at: https://blog.devgenius.io/10-useful-vscode-extensions-792c4eea51f [Accessed 3 Feb. 2024].

Chung, Thang. “How to Organize CLEAN ARCHITECTURE to Modular Patterns in 10 Minutes.” *HackerNoon.com*, 6 Nov. 2017, medium.com/hackernoon/applying-clean-architecture-on-web-application-with-modular-pattern-7b11f1b89011. Accessed 25 Jan. 2024.

Diehl, Ceci, et al. “Defining Recommendations to Guide User Interface Design: A Multimethod Approach (Preprint).” *JMIR Human Factors*, 10 Mar. 2022, https://doi.org/10.2196/37894.

“Effective Software Testing Strategies: A Guide for Quality Assurance.” *Www.linkedin.com*, www.linkedin.com/pulse/effective-software-testing-strategies-guide-quality-assurance. Accessed 25 Jan. 2024.

Garett, Renee, et al. “A Literature Review: Website Design and User Engagement.” *Online Journal of Communication and Media Technologies*, vol. 6, no. 3, July 2016, pp. 1–14. *NCBI*, www.ncbi.nlm.nih.gov/pmc/articles/PMC4974011/.

https://www.smashingmagazine.com/author/christopher-murphy. “A Comprehensive Guide to UI Design — Smashing Magazine.” *Smashing Magazine*, 2018, www.smashingmagazine.com/2018/02/comprehensive-guide-ui-design/.

Karsh, Patrick. “How Object-Oriented Design Enhances Code Readability and Maintainability.” *Medium*, 29 June 2023, medium.com/@patrickkarsh/how-object-oriented-design-enhances-code-readability-and-maintainability-e20d84a045ad. Accessed 25 Jan. 2024.

Square, Pepper. “Unlocking Engagement: The Power of UI/UX Design.” *Pepper Square*, 10 Jan. 2024, www.peppersquare.com/blog/ui-ux-in-user-engagement/. Accessed 25 Jan. 2024.

Tell, Disa, et al. “Lessons Learned from an Intersectoral Collaboration between the Public Sector, NGOs, and Sports Clubs to Meet the Needs of Vulnerable Youths.” *Societies*, vol. 12, no. 1, 20 Jan. 2022, p. 13, https://doi.org/10.3390/soc12010013.

“The Impact of UI/UX Design on User Engagement.” *Www.linkedin.com*, www.linkedin.com/pulse/impact-uiux-design-user-engagement-creating.

“UX Design: A Complete Guide on What It Is.” *Www.hotjar.com*, www.hotjar.com/ux-design/#:~:text=Simplicity%2C%20hierarchy%2C%20and%20consistency. Accessed 25 Jan. 2024.

Dautovic, A., Plosch, R. and Saft, M., 2011, July. Automatic checking of quality best practices in software development documents. In *2011 11th international conference on quality software* (pp. 208-217). IEEE.

Jiménez, R.C., Kuzak, M., Alhamdoosh, M., Barker, M., Batut, B., Borg, M., Capella-Gutierrez, S., Hong, N.C., Cook, M., Corpas, M. and Flannery, M., 2017. Four simple recommendations to encourage best practices in research software. *F1000Research*, *6*.